UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

ENTORNO DE PROGRAMACIÓN PARA APLICACIONES EN ROBÓTICA

William Mauricio Luguaña Chicharrón
Tutor: Guillermo González de Rivera Peces

MAYO 2016

ENTORNO DE PROGRAMACIÓN PARA APLICACIONES EN ROBÓTICA

AUTOR: William Mauricio Luguaña Chicharrón
TUTOR: Guillermo González de Rivera Peces

Human Computer Technology Lab
Dpto. Tecnología Electrónica y de Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2016

Resumen

En los últimos años, muchos hablan sobre las nuevas tecnologías, que están abriéndose paso en el ámbito educativo. Típicamente son ordenadores, tabletas, pizarras digitales, etc, pero actualmente han surgido nuevas tecnologías que muchos están empezando a resaltar: la programación como herramienta educativa.

Este Trabajo Fin de Grado se centra en la realización de una interfaz gráfica que permita la utilización de una plataforma hardware, básicamente un prototipo de robot, que sin entrar ni conocer el código, el usuario podrán seleccionar las funcionalidades que desee y se solicitaran los parámetros necesarios para la programación de los mismos. La aplicación podrá ser utilizada tanto por usuarios expertos como no expertos, a través de un sencillo entorno gráfico a diseñar. El objetivo final es disponer de un sistema completo de robótica educativa que pueda ser utilizado desde el colegio hasta la Universidad cuya finalidad será el aprendizaje de la robótica tanto si se parte desde el desconocimiento de la materia como si se maneja en los niveles más avanzados, donde el usuario podrá implementar las funciones que desee.

La aplicación tendrá disponible dos niveles de dificultad, básico y avanzado. En el nivel básico la interfaz permitirá, entre otras cosas, dar de alta o de baja los periféricos según la necesidad del usuario, cargar al robot las funcionalidades que el usuario haya seleccionado en el entorno gráfico, testear cada uno de los sensores y actuadores presentes en la plataforma hardware, mostrar la descripción en detalle de cada uno de estos periféricos y guardar, abrir el diseño gráfico elaborado por el usuario. En el nivel avanzado, el usuario podrá crear sus propias funciones en una librería y darlas de alta o de baja desde la aplicación para que ésta haga uso o no de las funciones que ha añadido o las que ya estén definidas.

Antes de llevar a cabo el análisis de los requisitos de la aplicación se ha realizado un estudio sobre el estado del arte, en la cual se pondrá en situación el proyecto respecto a otras tecnologías similares. En este apartado también se estudiará el protocolo de comunicación USB necesaria para establecer el intercambio de información entre el ordenador y el robot y viceversa.

Seguidamente se expondrá la solución del diseño tomado, la arquitectura software adoptada, descripción de los componentes que la forman y el protocolo de comunicación USB utilizado en nuestra aplicación.

Posteriormente viene una fase de desarrollo, en la cual se explicará las tecnologías y herramientas utilizadas para llevar a cabo la implementación de la interfaz gráfica.

Por último, se muestran las pruebas realizadas para verificar el funcionamiento de la interfaz gráfica y se incluyen unas conclusiones y trabajos futuros.

Palabras clave

Actuador, sensor, interfaz gráfica, robot, microprocesador, robótica, sistema, aplicación, protocolo de comunicación USB.

Abstract

In recent years, many talk about new technologies that are making their way in education. Typically they are computers, tablets, whiteboards, etc., but now new technologies have emerged that many are beginning to highlight: programming as an educational tool.

This final project focuses on the realization of a graphical interface that allows the use of a hardware platform, basically a prototype robot that without entering or know the code, the user can select the features you want and be requested parameters necessary for programming the same. The application can be used by both experts and non-experts, through a simple graphical design environment. The ultimate goal is to have a complete system of educational robotics that can be used from school to university whose purpose will be learning robotics whether you start from the ignorance of the matter as if handled in the most advanced levels, where the user can implement the features you want.

The application will be available two difficulty levels, basic and advanced. At the level of basic interface will allow, among other things, give high or low peripherals as needed by the user, load the robot functionalities that the user has selected in the graphical environment, test each of the sensors and actuators present in the hardware platform, showing the detailed description of each of these peripherals and save, open the graphic design developed by the user. At the advanced level, you can create your own functions in a bookstore and give them high or low from the application so that it uses or not the functions added or those that are already defined.

Before carrying out the analysis of application requirements it has conducted a study on the state of the art, in which we put into the project situation compared to other similar technologies. This section describes the necessary USB communication protocol will also look to establish the exchange of information between the computer and the robot and vice versa.

Then the design solution taken will be discussed, the software architecture adopted, description of the components that form and the USB communication protocol used in our application.

Later comes a development phase in which technologies and tools used to carry out the implementation of the GUI will be explained.

Finally, tests to verify operation of the GUI and conclusions and future work are included shown.

Keywords

Actuator, sensor, graphical interface, processor, microprocessor, robotics, system, application, USB communication protocol.

Agradecimientos

A la primera persona, que se lo quiero agradecer es a mi tutor Guillermo González de Rivera Peces, que sin su ayuda y sus conocimientos no hubiese sido capaz de realizar este trabajo.

Agradecer a mi familia, que me han apoyado en todo momento, especialmente a mis padres y hermano, por haberme proporcionado la mejor educación y lecciones de vida.

En especial a mi padre, por haberme enseñado que con esfuerzo, trabajo y constancia todo se consigue, y que en esta vida nadie regala nada.

En especial a mi madre, por cada día hacerme ver la vida de una forma diferente y confiar en mis decisiones.

A los compañeros de clase con los cuales he compartido grandes momentos.

Finalmente quiero agradecer a Ashley Carrasco por todo su apoyo demostrado en estos años de carrera, por la enorme paciencia que has tenido conmigo, gracias por hacerme creer que podía. También te agradezco por tranquilizarme y darme ánimos antes de los exámenes. Gracias por tu ayuda y apoyo que han sido muy importantes para poder finalizar como ingeniero esta carrera.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

En este apartado se proporcionar al lector una visión general de este trabajo, explicando la motivación, los objetivos planteados y la estructura del resto del documento.

1.1 Motivación

En los últimos tiempos han surgido muchas iniciativas cuyo objetivo es convertir la programación en una materia para introducir a niños en los fundamentos básicos de la programación de robótica sin necesidad de tener conocimiento de la misma. Un ejemplo de ello es la aplicación ArduBlock [1], es una interfaz gráfica sencilla que utiliza un sistema de bloques que simbolizan diferentes elementos de programación como: instrucciones, condiciones, variables, etc. Actualmente se pueden destacar dos plataformas las cuales son más populares: Google Blockly [2] (creada por Google) y Scratch [3] (desarrollada por el grupo Lifelong Kindergarten del MIT Media Lab).

En la actualidad el uso de estas aplicaciones empieza a abrirse hueco desde los colegios a universidades [4]. Ya hay colegios que imparten asignaturas de programación de robótica a los alumnos de Enseñanza Secundaria Obligatoria (ESO) [5]. Esta asignatura tiene como objetivo aumentar la motivación y fomentar la creatividad de los estudiantes.

Por lo tanto, la realización de este proyecto surge por la necesidad de implementar una interfaz gráfica que permita a estudiantes de colegios, institutos y universidades la utilización de una plataforma hardware (básicamente un prototipo de robot) dotada de sensores y actuadores, en la que se pueda seleccionar las funcionalidades que el usuario desee y los parámetros necesarios para la programación de los mismos. El hecho de que la aplicación este diseñada para la difusión desde los colegios a universidades se debe a que se podrá planificar tanto el aumento progresivo de dificultad y conocimientos, como el uso de una programación más avanzada en universidades o el uso libre para implementar funcionalidades que podrán ser añadidas a la funcionalidad inicial.

La realización de este TFG nace en paralelo con en el PFC ***"Plataforma modular de aprendizaje para robótica móvil"*** realizado por ***Jaime Díaz García*** [6] para poder implementar una plataforma de robótica educativa que disponga de una interfaz gráfica que pueda interactuar con un prototipo de robot. Por un lado, el PFC [6] se centrará en la realización del diseño y fabricación de un prototipo de robot dotado de un conjunto de periféricos. Su objetivo será la realización de la parte hardware y la implementación de una serie de rutinas para poner en funcionamiento el robot. Por otro lado, este TFG se centrará en la realización del software correspondiente a la parte de la interfaz gráfica.

Tengo que destacar que la implementación de las rutinas para poner en funcionamiento el robot se realizará por ambas partes para poder entender el funcionamiento de los diferentes periféricos de los que dispondrá el robot.

1.2 Objetivos

El objetivo principal de este proyecto es la realización de una interfaz gráfica, de manera que pueda ser utilizada tanto por usuarios expertos como no expertos. Como ya se ha indicado en apartados anteriores, el objetivo final es disponer de un sistema completo de robótica educativa que pueda ser utilizado desde el colegio hasta la Universidad.

La interfaz gráfica tendrá implementado un protocolo de comunicación USB PC-Robot y viceversa a través de la cual se enviarán/recibirán comandos, que han sido definidos previamente, para indicar al robot que acciones tiene que llevar a cabo. Del mismo modo, el robot también tendrá implementado un protocolo de comunicación para interpretar los comandos recibidos y enviar la respuesta a la interfaz gráfica.

La realización este proyecto implica familiarizarse con las distintas herramientas de protocolo de comunicación USB, el uso de software de programación, compiladores de microcontroladores y el manejo de interfaces gráficas en java.

Para alcanzar dicho propósito, se plantean los siguientes objetivos:

- Permitir al usuario consultar los periféricos que tiene el robot.
- Ver una descripción de cada uno de estos periféricos.
- Realizar un testeo de cada uno de estos periféricos. Para este apartado será necesario establecer una comunicación PC-Robot y viceversa.
- Permitir al usuario crear un diseño gráfico al cual irá añadiendo los periféricos disponibles en el robot.
- Seleccionar las funcionalidades que va a llevar a cabo el robot. La definición de estas funcionalidades dependerá de los periféricos añadidos al diseño gráfico.
- Cargar al robot las funcionalidades definidas por el usuario a través del protocolo de comunicación.
- Dar de alta o de baja los periféricos que se desee, para posteriormente poder usarlos o no en la interfaz gráfica.
- La aplicación dispondrá de un nivel de dificultad avanzado. En este nivel, además de lo anterior, el usuario podrá dar de alta o de baja las posibles situaciones en la que se podrá encontrar el robot, y las acciones que llevará a cabo en tales situaciones. Estas situaciones y acciones estarán asociadas a un periférico.
- En este nivel de dificultad, el usuario dispondrá de una librería elaborada en lenguaje de programación C en la que podrá ir añadiendo sus propias funciones que posteriormente podrá darlas de alta o de baja desde la interfaz gráfica.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Introducción:** En este apartado se exponen las motivaciones que han llevado a la realización de este trabajo y los objetivos del mismo.

- **Estado del arte:** Aquí se hablará de los aspectos que son importantes en este TFG. Se realizará un estudio de las distintas tecnologías investigadas, valoradas y estudiadas como posible solución.
  También se realizará un estudio de evolución de la programación de herramientas educativas hasta la actualidad, para ver la importancia de este campo. Finalmente se hace hincapié en el funcionamiento del protocolo de comunicación USB.

- **Análisis de los requisitos:** En este apartado se dará una descripción general del proyecto, las funcionalidades que tiene que cumplir y los objetivos esperados. También se centrará en el catálogo de requisitos del software y se describirán los casos de usos más relevantes.

- **Diseño:** En este apartado se expondrá la solución de diseño tomada, la arquitectura software adoptada, una descripción de los componentes que la forman y el protocolo de comunicación USB utilizado en nuestro software.

- **Desarrollo:** En esta fase se explicará la interfaz gráfica creada para el control del robot. Este apartado es muy visual ya que se explica uno por uno cómo funciona cada botón de la interfaz gráfica y las pantallas que van surgiendo cuando se maneja la aplicación. Para ello se comenzará describiendo las plataformas y tecnologías empleadas, se detallará la implementación de cada componente de la arquitectura software, se describirán los paquetes y librerías utilizados, los protocolos de comunicación usados tanto en la parte del robot como en la aplicación y finalizando con la interfaz de usuario.

- **Integración, pruebas y resultados:** Se expone las pruebas realizadas sobre todo el diseño y se presentará los resultados obtenidos.

- **Conclusiones y trabajo futuro:** En este apartado se expone las conclusiones extraídas a lo largo del desarrollo, así como las posibles mejoras y modificaciones que se pueden hacer y futuros trabajos a realizar.

- **Referencias:** Bibliografía utilizada para la realización de este trabajo.

- **Glosario**

- **Anexos:**
  - **Descripción de los Casos de Uso**
  - **Requisitos funcionales del Subsistema Usuario**
  - **Descripción de las funciones de la API**
  - **Diseño conceptual de las ventanas**
  - **Código funciones más relevantes**
  - **Manual de Programación USB DFU Bootloader**
  - **Manual instalación controlador LUFA CDC Demo**
  - **Interfaz de usuario**
  - **Tabla comandos**
  - **Pruebas de caja negra**
  - **Pruebas del sistema completo**
  - **Pruebas de caja blanca**

2 Estado del arte

2.1 Introducción

En este apartado se explicará el contexto tecnológico en el cual se ha centrado este trabajo. Se estudiarán y analizarán las distintas posibilidades para llevar a cabos las tareas implicadas. También se realizará un estudio de evolución de la programación de aplicaciones educativas, y se terminará con el estudio del protocolo de comunicación USB para el intercambio de información entre el ordenador y el robot de forma bidireccional.

Dado que el presente proyecto tienen como objetivo diseñar una aplicación cuya finalidad es el aprendizaje de la robótica, se empezara este apartado definiendo el término: Robótica.

2.1.1 Robótica

La robótica es la rama de la ingeniería mecánica, eléctrica y electrónica que se ocupa del diseño y construcción de robots. Los robots son máquinas controladas por el ordenador y programadas para manipular objetos, moverse y realizar acciones al mismo tiempo que interactúan con su entorno.

Ilustración 1: Robot Dot and Dash

Por lo tanto, el robot se puede considerar como una máquina dotada de un ordenador o como un ordenador con dispositivos de entrada y salida, que dispondrá de un programa cargado en memoria cuyo objetivo es realizar las acciones indicadas en el programa para dirigir sus movimientos. Si el usuario tiene la necesidad de indicar al robot que realice nuevas acciones, será necesario cambiar el programa añadiéndole la nueva funcionalidad.

2.1.2 Robótica y programación educativa

La robótica educativa es una herramienta de aprendizaje que puede describirse como un proceso sistemático y organizado, cuyo objetivo final es la generación de entornos de aprendizaje de robótica, en los que el estudiante pueda concebir, desarrollar y poner en práctica diferentes ideas que le permita resolver ciertos problemas.

El alto nivel de abstracción y poder entender la complejidad de los conceptos, es un impedimento para muchas personas ya que el nivel de dificultad es grande. Aprender a programar es algo que todos deberíamos conocer, no solo aquellos que lo han aprendido como oficio. Gracias al aprendizaje de la programación se puede aprender a crear nuestras propias herramientas y cambiar nuestra manera de pensar y de ver el mundo que nos rodea.

Si se quiere inducir a un estudiante al aprendizaje de la programación, está claro que no se va a empezar hablándoles de lenguajes de programación, variables, algoritmos y otras palabrejas relacionadas con programar. Con lo cual, es mejor empezar con algo más fácil a lo que puedan habituarse y que puedan manejar por sí mismo. Actualmente existen gran variedad de herramientas enfocados a la enseñanza de programación para niños que permiten crear programas como si fuera un puzzle [7]. Entre éstas, existen herramientas diseñadas especialmente para que los estudiantes puedan aprender robótica, programación y electrónica con la plataforma de Arduino [8]. Estas herramientas sirven de gran ayuda para que los estudiantes puedan empezar con buen pie en el amplio mundo de la programación. En uno de los apartados siguientes se repasa algunas herramientas de programación educativa.

2.1.2.1 Ambientes de aprendizaje con la robótica

En los últimos tiempos se han ido desarrollando plataformas robóticas con equipos de hardware y software instalados con un fin específico y que posibilitan el desarrollo de entornos de aprendizaje cuya objetivo es la construcción y programación del robot. El desarrollo de plataformas robóticas a bajo costo fortalece el aprendizaje de diferentes áreas de conocimiento que permiten su adquisición por las instituciones educativas y, con estas, la diversificación del tipo de experiencias que son ejecutadas en el aula.

Estas plataformas robóticas son utilizadas con propósitos formativos y algunas de estas han sido creadas para los diferentes niveles escolares y otras para trabajar áreas específicas de conocimiento, pero el éxito de su utilización depende de la formación que se dé a los docentes para la planeación y ejecución de actividades que las incluyan en la construcción de conocimiento en las diversas temáticas del área en estudio.

2.1.3 Plataformas de programación de robótica educativa

A continuación se citan las principales plataformas que son utilizadas para programar las funcionalidades necesarias para poder utilizar y controlar un robot.

2.1.3.1 Dot and Dash [9]

Son robots educativos que impulsan el aprendizaje de estudiantes de entre 5 y 14 años. Han sido diseñados para potenciar las habilidades e introducir a los niños en el mundo de la tecnología y la programación [10].

Ilustración 2: Robot Dot and Dash

Estos robots pueden ser programados a partir de una App de programación que simula un juego de bloques desarrollado especialmente para niños. La app se llama Blockly [11] que permite al estudiante aprender conceptos de programación desde muy temprana edad, al construirse como piezas de un rompecabezas: desde comandos simples hasta secuencias complejas. La programación en ambos robots se controla desde una Tableta Android o Apple.

Ilustración 3: Aplicación para programar el robot Dot and Dash (Blockly)

2.1.3.2 mBlock Robot Educativo [12]

Es una aplicación ideal de robótica educativa basada en el software Scratch 2.0 [13] para que escuelas y centros de formación pueda introducir la robótica de forma sencilla y enseñar a programar robots basados en Arduino [14]. Este software está especialmente diseñado para niños, con el cual, podrán controlar su robot mediante el método de arrastrar y soltar elementos.

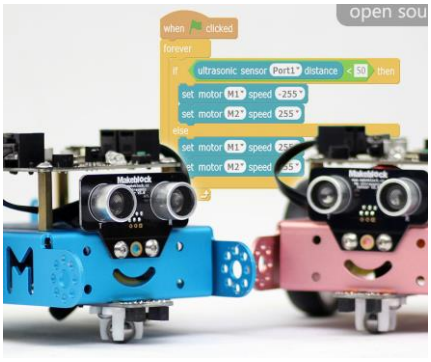

Ilustración 4: mBloq Robot educativo

2.1.3.3 ArduBlock [15]

Es una herramienta que acerca el mundo de la programación de Arduino a estudiantes y aficionados de una forma fácil y sencilla. Su interfaz gráfica utiliza un sistema de bloques, los cuales hacen referencia a diferentes componentes de programación, como: instrucciones, condiciones, variables, etc. Del mismo modo que en las plataformas anteriores, los bloques de programación se van uniendo como piezas de puzzle hasta formar un programa. De esta forma, se puede desarrollar programas sin necesidad de escribir código.

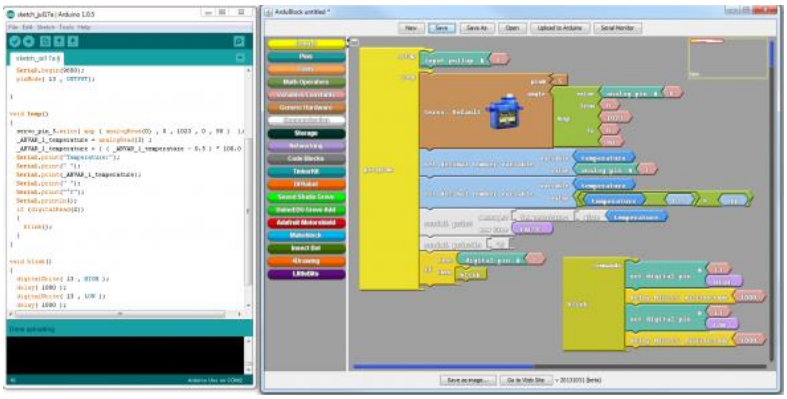

Ilustración 5: Herramienta para la programación de Arduino

2.1.3.4 S4A [16]

Es una herramienta que permite programar la plataforma de hardware libre Arduino de una formar fácil y sencilla. Esta herramienta es una modificación de Scratch [13], incluyendo bloques nuevos para tratar los posibles sensores y actuadores conectados a una placa de Arduino [14]. Los bloques de programación se van uniendo como piezas de puzzle hasta formar un programa, así los usuarios podrán desarrollar programas sin necesidad de escribir código.

Su objetivo es atraer a gente al mundo de la programación y proporcionar a los programadores de Arduino [14] una interfaz de alto nivel.

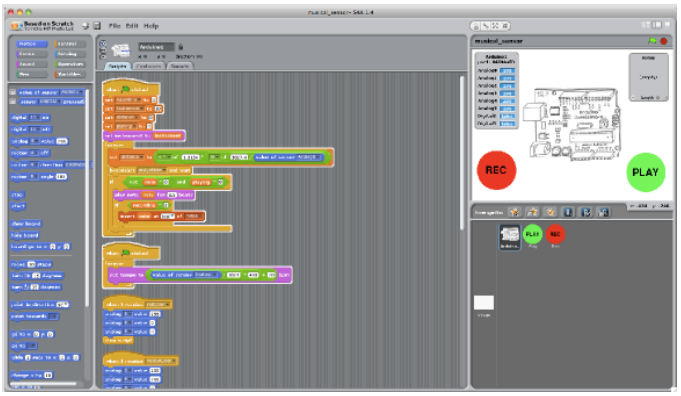

Ilustración 6: S4A plataforma para programar una placa de Arduino

2.1.4 Protocolo de comunicación USB

2.1.4.1 Introducción

Los grandes sistemas automáticos y los ordenadores necesitan un medio de comunicación a través de la cual intercambiar información y órdenes con otros dispositivos.

La comunicación USB, que se va a tratar en este apartado, es utilizado para establecer comunicación entre un ordenador y un dispositivo externo que en este caso es un robot.

En la interfaz gráfica desarrollada en este trabajo la comunicación entre el ordenador y robot es importante, para el envío de datos sobre un canal de comunicación o un bus. Por lo tanto, en este se va a estudiar el protocolo de comunicación USB y su funcionalidad.

2.1.4.2 Protocolo USB

El Bus Universal en Serie, más conocido por la sigla USB, es un bus estándar industrial que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos [17].

Ilustración 7: Icono del dispositivo USB

El USB es utilizado como un estándar de conexión de periféricos como: teclados, ratones, memoria USB, impresoras, etc. Fue originalmente pensado para conectar dispositivos a los ordenadores eliminando la necesidad de conectar tarjetas PCI y también para conectar y desconectar dispositivos sin tener que reiniciar el pc. Actualmente ha desplazado a los puertos serie, paralelo, puerto de juegos, etc del mercado o a la simple consideración de eliminar estos dispositivos obsoletos de los ordenadores modernos.

2.1.4.3 Topología

El USB, se caracteriza por tener conexión punto a punto. La conexión parte desde un host, que puede ser un PC o hub, hasta un periférico u otro hub. Un hub es un dispositivo que permite concentrar varios puertos USB, permitiendo la conexión con una máquina mediante un solo bus o cable [18].

Ilustración 8: Dispositivo Hub

2.1.4.4 Funcionamiento

Los dispositivos tienen asociados unos canales lógicos unidireccionales llamados pipes que conectan al host controlador con una entidad lógica en el dispositivo llamado endpoint. Los datos son enviados en paquetes de longitud variables de 64, 128 o más bytes.

Cuando un dispositivo es conectado al puerto USB, el host controlador le asigna una dirección única de 7 bits, que es utilizada para identificar el dispositivo en la comunicación. Continuamente, el host controlador está consultando a los dispositivos para ver si tiene algo para mandar, de esta forma ningún dispositivo puede enviar datos sin solicitud previa explícita del host controlador.

Para acceder a un endpoint se utiliza una configuración jerárquica de la siguiente manera:

- Un dispositivo conectado al bus tiene un único descriptor de dispositivo, quien a su vez tiene uno o varios descriptores de configuración.
- Estos descriptores guardan el estado del dispositivo.
- Cada descriptor de configuración tiene uno o varios descriptores de interfaz.
- Los descriptores de interfaz tienen una configuración por defecto, que son los que tienen los endpoints, que a su vez pueden ser reutilizadas entre varias interfaces.

2.1.4.5 Tipos USB

Los dispositivos USB se pueden clasificar por su velocidad de transmisión de datos:

- **Velocidad baja (1.0):** tasa de transferencia de hasta 1,5Mbit/s, utilizado por los dispositivos HID (Human Interface Device) como teclados, ratones, etc.
- **Velocidad completa (1.1):** tasa de transferencia de hasta 12Mbit/s. Estos dispositivos dividen el ancho de banda de la conexión USB entre ellos.
- **Alta velocidad (2.0):** tasa de transferencia de hasta 480Mbit/s cuya tasa real es de 280Mbit/s. Dispone de cuatro líneas un par para datos y otro para la alimentación.
- **Superalta velocidad (3.0):** tasa de transferencia de hasta 4Gbit/s. Compatible con los estándares anteriores.

2.1.4.6 Señalización y conectores

Las señales USB son transmitidas por un par trenzado cuyos hilos son denominados D+ y D- utilizando señalización half-duplex, minimizando el ruido electromagnético en tramos largo.

Hay tres tipos de conectores: estándar, mini y micro. Los estándares son los que típicamente se encuentran en el ordenador y vienen en dos tipos: A y B. El tipo A es plano y se encuentra en el lado del host controlador y el B es el cuadrado y se encuentra del lado del dispositivo. Hay que decir que todos los cables son machos, mientras que los enchufes, ya sea del ordenador o dispositivo, son hembras.

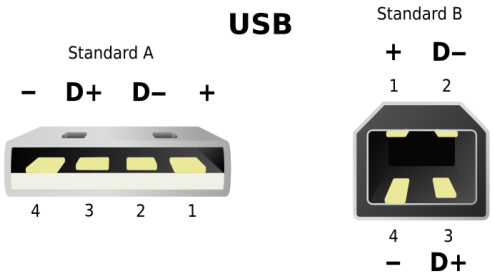

Ilustración 9: Conector estándar A y B

2.1.4.7 Aplicaciones USB

Para desarrollar una aplicación USB se necesita:

- Un microcontrolador que soporte la interfaz USB.
- Un programa sobre el periférico elaborado en cualquier lenguaje de programación para transmitir información.
- Un ordenador con puerto USB.
- Herramientas para la implementación de programas para los microcontroladores.

2.1.4.8 Protocolo USB CDC

La clase de dispositivo de comunicación (CDC), es compatible con una amplia gama de dispositivos que pueden realizar funciones de telecomunicaciones y redes. Esta clase simplifica al usuario la comunicación USB proporcionando una funcionalidad de puerto COM [19]. El CDC proporciona una sencilla UART para trasmitir y recibir datos a y desde el pc.

2.1.4.8.1 Características de la CDC

Un dispositivo de comunicación tiene tres tareas básicas:

- La gestión de dispositivos: controlar la configuración de un dispositivo específico y notificar al host USB de ciertos eventos.
- La gestión de llamadas: establecer y terminar las llamadas telefónicas u otras conexiones.
- La transmisión de datos: enviar y recibir datos de aplicaciones.

La implementación CDC en los componentes USB tienen las siguientes características:

- Emular un puerto COM virtual utilizando la ACM (Modelo de control Abstracto) subclase de CDC.
- Emular un adaptador Ethernet utilizando NMC (Modelo de RED de control) subclase de los CDC.
- El dispositivo USB soporta la clase CDC (ACM) para aplicaciones de dispositivos USB y USB Host.

3 Análisis

3.1 Descripción general

3.1.1 Propósito del sistema

La aplicación desarrollada en este trabajo estará orientada a la programación de la robótica que servirá a los usuarios para crear diseños, en un entorno gráfico, dotados de sensores y actuadores que hayan sido definidos previamente. El diseño creado por el usuario será previamente cargado en el robot desarrollado en el ***PFC [6]***.

La programación del diseño en el robot se realizara a través de la comunicación USB de forma transparente para el usuario. Una vez que el usuario haya definido los sensores y actuadores, en el diseño, tendrá que seleccionar funciones que desea que realice el robot. El usuario no tiene conocimiento de cómo se realiza la programación de dichas funciones. Cuando el usuario carga su diseño con las funcionalidades que ha definido, el sistema genera de forma automática el programa que previamente se cargará en el robot.

El sistema dispondrá de un nivel de dificultad avanzado, donde las funciones que puede llevar a cabo el robot pueden ser ampliadas por el usuario, es decir, aquellos usuarios que tengan conocimiento de la programación de microcontroladores dispondrá de una librería elaborada en lenguaje de programación C en la que podrá ir añadiendo sus propias funciones que posteriormente podrá darlas de alta o de baja desde la aplicación.

Las funcionalidades que va a llevar a cabo el robot son combinaciones de una **situación** en el que se encuentra el robot y la **acción** que llevará acabo en tal situación. Con lo cual, cuando el usuario va a dar de alta una situación desde la aplicación, deberá seleccionar el periférico, introducir la situación en la que se encontrará el robot y la función correspondiente a dicha situación. Por otro lado, si desea dar de alta una acción deberá seleccionar el periférico, introducir la acción que realizará el robot y la función correspondiente a dicha acción.

En ambos niveles de dificultad, la aplicación permitirá al usuario guardar el diseño que haya elaborado, cargar un diseño que haya guardado previamente, realizar un testeo de cada uno de los periféricos presentes en la aplicación y dar de alta o de baja ciertos periféricos para posteriormente poder usarlos en la interfaz gráfica.

3.1.2 Ámbito del sistema

La aplicación solo permitirá crear diseños en los cuales hará uso de aquellos periféricos que se hayan definido en el ***PFC [6]***, por lo que queda fuera del alcance del sistema la incorporación de nuevos periféricos sobre en la aplicación.

Desde una primera perspectiva general sobre la aplicación, se puede describir todas aquellas acciones que los usuarios podrán llevar a cabo de la siguiente manera:

- Seleccionar el puerto de comunicación con el robot.
- Cargar los periféricos que dispone el robot.
- Realizar un testeo de los periféricos que dispone el robot.
- Consultar la información de cada periférico disponible en el robot.
- Crear un diseño añadiendo los sensores y actuadores que se desee.
- Seleccionar las posibles situaciones en la que se puede encontrar el robot.
- Seleccionar la acción que se desea que realice el robot en una situación.
- Cargar toda la funcionalidad seleccionada en el robot.
- Dar de alta o de baja ciertos periféricos para posteriormente usarlos o no en la aplicación.
- Además de lo anterior, en el nivel de dificultad avanzado el usuario podrá dar de alta o de baja situaciones y acciones definidas previamente en la librería elaborada en lenguaje de programación C.

3.2 Modelo de Casos de usos

El objetivo de este apartado es detallar el modelo de casos de uso de la aplicación. Los casos de uso son descripciones detalladas del proceso de manifestación de cierta funcionalidad del software frente a un cierto evento provocado por un actor. Utilizamos los casos de uso para describir el uso del sistema y como los usuarios interactúan con nuestra aplicación.

3.2.1 Definición de los actores

En este apartado se enumeran los actores que intervienen en el sistema y se dará una breve descripción de los mismos. Los actores son todas aquellas entidades externas que pueden interactuar con la aplicación. En nuestro caso dicha entidad será el Usuario, con lo cual, la funcionalidad del sistema se descompondrá en un actor: Usuario. A continuación se describirán en rasgos generales este actor.

3.2.1.1 Usuario

Este actor representa al usuario del sistema quien se encarga de ejecutar la aplicación y es quien utiliza el programa. Este actor contiene toda la funcionalidad relacionada directamente con el usuario del sistema, y agrupará los requisitos que definan restricciones que afectan de forma directa al propio usuario.

3.2.2 Detalle de los casos de uso

En este subapartado se define los casos de uso más representativos del actor Usuario. En el Anexo A se detalla cada caso de uso.

Casos de Uso de Usuario:

- **Seleccionar puerto de comunicación [UC-0001]:** el Usuario selecciona el puerto de comunicación con el Robot.
- **Cargar periférico [UC-0002]:** el Usuario que quiere cargar los periféricos disponibles del robot.
- **Ver detalle periférico [UC-0003]:** el Usuario que quiere consultar la descripción de cada uno de los periféricos del robot.
- **Testeo de los periféricos [UC-0004]:** el Usuario que quiere testear cada uno de los periféricos del robot.
- **Añadir periférico [UC-0005]:** el Usuario que quiere añadir un periférico al diseño elaborado en el entorno gráfico.
- **Eliminar periférico [UC-0006]:** el Usuario que quiere eliminar un periférico del diseño elaborado en el entorno gráfico.
- **Eliminar todos los periféricos [UC-0007]:** el Usuario que quiere eliminar todos los periféricos del diseño elaborado en el entorno gráfico.
- **Seleccionar situación [UC-0008]:** el Usuario que quiere seleccionar la situación en la que se encuentra el robot.
- **Seleccionar acción [UC-0009]:** el Usuario que quiere seleccionar las acciones que quiere que realice el robot en una situación.
- **Cargar funcionalidad al robot [UC-0010]:** el Usuario quiere cargar al robot la funcionalidad obtenida como las combinaciones de la situación y acción seleccionada.
- **Guardar diseño [UC-0011]:** el Usuario que quiere guardar el diseño elaborado en el entorno gráfico.
- **Abrir diseño [UC-0012]:** el Usuario que quiere abrir un diseño guardado previamente.
- **Dar de alta un periférico [UC-0013]:** el Usuario que quiere añadir un nuevo periférico para poderlo usar en la aplicación.
- **Dar de baja un periférico [UC-0014]:** el Usuario que quiere eliminar un periférico de la aplicación.
- **Dar de alta una situación del robot [UC-0015]:** el Usuario que quiere dar de alta un periférico con una nueva situación en la que se podrá encontrar el robot.
- **Dar de baja una situación del robot [UC-0016]:** el Usuario que quiere eliminar de la aplicación la situación correspondiente a un periférico del robot.
- **Dar de alta una acción [UC-0017]:** el Usuario que quiere dar de alta un periférico con una nueva acción que realizará el robot.
- **Dar de baja una acción [UC-0018]:** el Usuario que quiere eliminar de la aplicación la acción correspondiente a un periférico del robot.

3.3 Catálogos de requisitos

En este apartado, se desarrolla las especificaciones del comportamiento del sistema. Contiene los requisitos funcionales y no funcionales.

3.3.1 Requisitos funcionales

En esta sección se escribirán los requisitos deducidos a partir de la idea inicial del sistema y del proceso *brainstorming* realizado. Los requisitos funcionales constituye la declaración de los servicios que el sistema debe proporcionar, como debe reaccionar ante una entrada particular y cómo se debe comportar ante situaciones particulares. En general, describe las funcionalidades de que debe disponer el sistema y de qué forma va a utilizarlo el usuario.

En el Anexo B detallo los requisitos funcionales encontrados para el sistema. En cada uno de ellos trato de expresar de forma clara y con exactitud los servicios que el sistema debe proporcionar y como debe comportarse ante situaciones particulares.

3.3.2 Requisitos no funcionales

Los requisitos no funcionales definen las características y restricciones sobre cómo debe actuar el sistema.

En los subapartados siguientes se detallan los requisitos no funcionales encontrados para el sistema.

3.3.2.1 Lenguaje de programación

El sistema estará programado en el entorno de desarrollo NetBeans y en el lenguaje JAVA.

3.3.2.2 Usabilidad

El sistema deberá facilitar el uso de la aplicación, con el objetivo que éste pueda ser utilizado por personal no especializado. Por lo tanto, la aplicación estará dirigida a usuarios de distinta edad y sin conocimiento de la programación, por lo que se pretende que la aplicación sea usable, intuitiva y sencilla de utilizar.

3.3.2.3 Plataformas

El sistema deberá ser accesible desde cualquier ordenador independiente del sistema operativo que use.

3.3.2.4 Escalabilidad

El sistema será fácilmente escalable con el fin de poder introducir mejoras futuras con la mayor facilidad posible y sin tener que rediseñar gran parte del sistema.

3.3.2.5 Rendimiento

Se requiere que las operaciones más costosas del sistema sean ejecutadas con la mayor rapidez posible. Esto garantizará una mayor fluidez de la aplicación e incrementará la sensación de retroalimentación al usuario.

4 Diseño

4.1 Introducción

Tras haber presentado el análisis del sistema en el apartado anterior, a continuación se expondrá el diseño de la aplicación indicando el contexto en el cual se empleará el sistema, la arquitectura y las funcionalidades que dispondrá la aplicación. Una vez descrito esto, se definirán los componentes que formarán parte de la aplicación. También se dará una idea preliminar de la interfaz gráfica que se va a implementar.

4.1.1 Contexto del sistema

En este subapartado se da una visión general de sistema, ver el entorno en que se va a utilizar y sus límites. En este caso, los usuarios que interactuarán con la aplicación son generalmente estudiantes cuyo papel es elaborar un diseño, en un entorno gráfico, al cual irán añadiendo los periféricos que deseen. Una vez hayan añadido los periféricos deseados, la aplicación proporcionará al usuario las posibles situaciones en la que se podrá encontrar el robot y las acciones que podrá llevar a cabo los periféricos en tales situaciones. Estas situaciones y acciones se cargan dinámicamente dependiendo de los periféricos definidos en su diseño. Tras definir las funcionalidades que se quiere que realice el robot, el usuario podrá cargarlas sobre la plataforma hardware. Para poder cargar estas funciones será necesario establecer comunicación con el robot.

Esta comunicación también se utilizará para enviar y recibir comandos a y desde el robot, el cual dependiendo del comando recibido realizará una función u otra.

Tal y como se ha mencionado anteriormente, la aplicación mostrará al usuario las posibles situaciones y acciones que dispondrá el robot. Estas situaciones y acciones serán obtenidas de los ficheros que se muestran a continuación:

- Situaciones.txt → en este fichero se almacenarán las posibles situaciones en la que se puede encontrar el robot.
- Acciones.txt → en este fichero almacenará las acciones que puede realizar el robot en cualquiera de las situaciones definidas en el primer fichero.

Las situaciones y acciones definidas en los ficheros son ampliables, es decir, aquellos estudiantes que tengan idea de programación de microcontroladores podrán elaborar nuevas funciones que se añadirán a una librería elaborada en lenguaje de programación C y que posteriormente podrán darlas de alta o de baja desde la interfaz gráfica. Para dar de alta o de baja estas funcionalidades, es necesario que el usuario haya seleccionado el nivel de dificultad avanzado.

4.1.2 Arquitectura del Software

La arquitectura del software de un sistema o de un programa describe la aplicación en términos de su organización conceptual en capas, paquetes, frameworks, clase, interfaces y subsistemas. Este apartado se centrará en la definición de los componentes y en la interacción de los mismos. Hay que destacar que la aplicación desarrollada cuenta con tres partes bien diferenciadas.

- Interfaz de interacción con el usuario
- Los datos (situaciones, acciones y periféricos del robot) almacenados en tres ficheros de texto.
- La lógica de unión entre las dos partes anteriores.

Con esta definición se puede deducir que el patrón de diseño más adecuado que podemos emplear es el conocido como Modelo Vista Controlador (MVC). Este patrón de diseño MVC separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

En la Ilustración 10 se muestra los componentes del sistema, que sigue el patrón de diseño MVC, así como la secuencia de acciones que se producirán en una interacción entre el usuario y la aplicación.

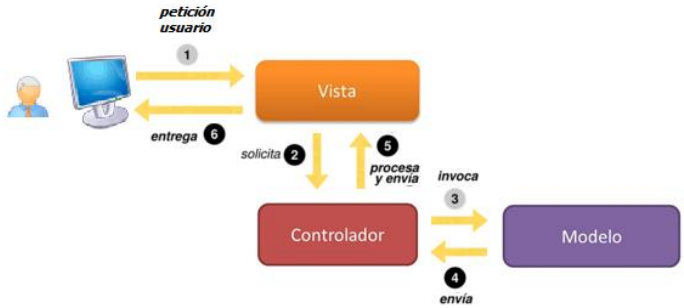

Ilustración 10: Arquitectura MVC del sistema

Tal y como se puede ver, este modelo tiene tres capas que pasaremos a describir brevemente:

- **Modelo:** esta capa contiene los datos almacenados con los que trabaja la aplicación y para permitir interactuar con ellos y representarlos.
- **Vista:** esta capa es la interfaz que utiliza el usuario para interactuar con la aplicación y a través de la cual se representan los datos.
- **Controlador:** esta capa representa el papel mediador entre las dos capas anteriores. Principalmente recibe peticiones de la Vista, que vienen del usuario, y realiza los cambios oportunos en el Modelo. Finaliza refrescando la Vista con estos cambios para que se muestre al usuario.

El proceso de funcionamiento del modelo anterior es el siguiente:

En primer lugar, el usuario envía una petición a la vista. El controlador recibe las peticiones del usuario a través de la Vista pero no realiza ninguna lógica de navegación, simplemente utiliza el modelo de datos y la capa de acceso a los datos para cargar todos los datos pertinentes en el marco de ejecución. Por lo tanto, el controlador se encargará de realizar las operaciones adecuadas en el modelo de datos y generará una respuesta que será devuelto a la vista.

El modelo de datos contiene propiedades, métodos y eventos para encapsular el estado de la vista y la lógica de interacción. El modelo devolverá los datos solicitados por el controlador y éste seleccionará una vista que posteriormente la devolverá al usuario con todos los datos seleccionados del modelo.

Se puede observar que la única capa que actúa con el modelo es el controlador y la capa que tiene contacto con el usuario final es la vista que contiene la interfaz gráfica. Cuando el usuario interactúa con la interfaz, la información que introduce en la aplicación, a través de ciertos eventos: pulsaciones de botones o datos introducidos, etc., debe comunicarse al controlador, el cual será el encargado de notificar los cambios al modelo para que realice los cambios oportunos. A continuación se expone un diagrama en la que se ilustra todo este proceso:

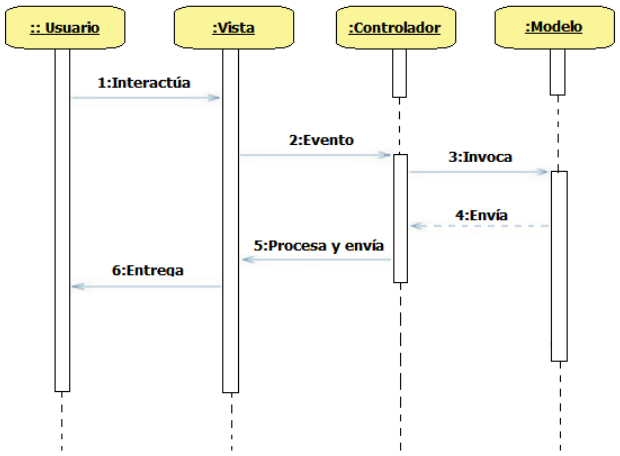

Ilustración 11: Comunicación Modelo, Vista, Controlador

Creemos que es muy interesante tratar en profundidad los distintos componentes del MVC:

- **El Modelo:** el modelo representa los datos y la información con la que trabajaremos en la aplicación. Es el primer componente funcional e independiente de nuestro sistema.
  El modelo resulta ser el almacén de la información, pero no las acciones o eventos que las manipulan., que normalmente se acaba diseñando sobre una base de datos. En nuestro caso, el modelo de datos son ficheros de texto exactamente tres. Uno **Situaciones.txt** que como ya se ha comentado anteriormente, almacena las posibles situaciones en la que se puede encontrar el robot. El otro fichero es **Acciones.txt**, que almacena las posibles acciones que puede llevar a cabo el robot en cualquiera de las situaciones definidas. Por último está el fichero **Periféricos.txt,** que almacena los periféricos de los que

dispondrá nuestra aplicación. Los tres ficheros pueden ser modificados de tal forma que el usuario pueda dar de alta o de baja las situaciones, acciones o periféricos desde la aplicación.

- **Vista:** la vista es la que ve el usuario y con la que estamos más familiarizados. Básicamente constituirá la interfaz gráfica que utilizará el controlador para acceder a los datos y presentarlos al usuario.

  Esta capa tiene como objetivo principal mostrar los datos de la aplicación al usuario y ofrecer comportamientos, eventos y enlaces a datos para que el usuario pueda interactuar con el sistema.

- **Controlador:** es la capa que contiene el conjunto de funciones de acceso a la información para poder leer los datos almacenados en nuestro modelo y mostrarlos al usuario. Hay que dejar claro que estas funciones no manipulan los datos directamente, esta tarea está asignada al modelo, que dispondrá de las funciones necesarias para poder manipular los datos. En nuestro caso, son todas aquellas funciones que componen la interfaz de acceso a los datos.

4.1.3 Estructura del Modelo de Datos *(Modelo)*

Nuestro modelo de datos estará compuesto por una serie de ficheros en los cuales se almacenará todos los datos que dispondrá el sistema y que serán manipulados por el controlador. Es uno de los puntos más importantes para que la aplicación pueda funcionar de forma correcta ya que estos ficheros almacenan la información más relevante de la aplicación.

A continuación se describen los ficheros en los cuales se almacenará todos los datos que manipulará la aplicación. Básicamente son tres: Periféricos.txt, Situaciones.txt y Acciones.txt.

4.1.3.1 Periféricos.txt

Este fichero, en un principio, estará compuesto por todos los periféricos que se ha definido en el ***PFC [6]***, pero posteriormente se podrá ir añadiendo o quitando los periféricos según la necesidad del usuario. Para realizar estas operaciones, la interfaz gráfica cuenta con la funcionalidad para poder dar de alta o de baja los periféricos que posteriormente se presentarán o no en la aplicación.

La estructura que tendrá este fichero es la siguiente:

- La primera línea del fichero corresponderá a una cabecera, en la cual se indicará el periférico y el número máximo de este periférico en el robot.
- Las líneas siguientes estarán compuestas por los periféricos que el usuario haya ido dando de alta.

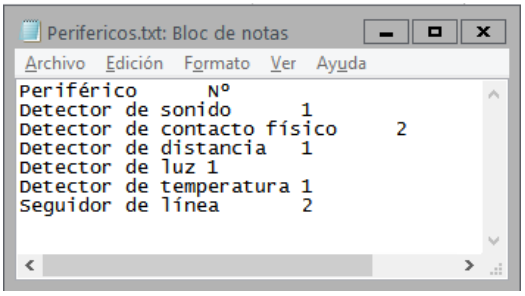

Ilustración 12: Fichero Periféricos.txt

Nuestra aplicación hará uso de este fichero cuando el usuario inicia la aplicación y tiene la necesidad de cargar los periféricos en la interfaz gráfica para poder empezar a trabajar con ellos en el entorno gráfico así como para realizar el testeo y ver la descripción de cada uno de ellos.

4.1.3.2 Situaciones.txt

Este fichero almacenará las posibles situaciones en la que se puede encontrar el robot. Inicialmente contará con un par de situaciones, con su correspondiente función, para algunos de los periféricos definidos en el ***PFC [6]***. Estas situaciones podrán ser ampliadas cuando el usuario este trabajando con la aplicación en el nivel de dificultad Avanzado, en la cual podrán añadir o eliminar las situaciones según su necesidad. Para realizar estas operaciones, la interfaz gráfica cuenta con la funcionalidad para poder dar de alta o de baja las situaciones correspondientes a un periférico. Hay que destacar que cada situación estará ligada a una función que ha sido previamente definida en la API o librería C que se ha elaborada para gestionar las posibles funcionalidades que podrá realizar el robot.

La estructura que tendrá este fichero es la siguiente:

- La primera línea del fichero corresponderá a una cabecera, en la cual se indicará:
  - **Periférico:** definirá el nombre del sensor o del actuador.
  - **I/D:** Para identificar el número de periféricos en el robot.
  - **Situaciones:** son las diferentes situaciones en la que se podrá encontrar el robot.
  - **Función:** es la función que se ha añadido a la API o librería y la cual corresponde a la situación que se va a dar de alta en la aplicación.

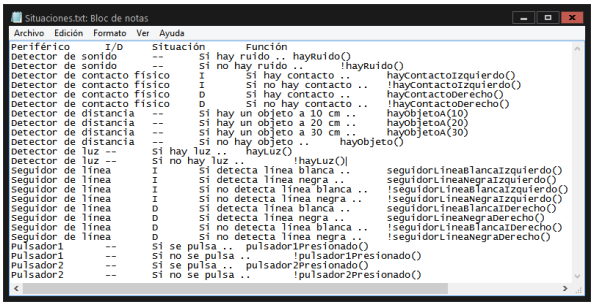

Ilustración 13: Fichero Situaciones.txt

- Las líneas siguientes estarán compuestas por los distintos periféricos con las situaciones que ya estuviesen definidas o que hayan sido dadas de alta por el usuario.

Nuestra aplicación hará uso de este fichero cuando el usuario, una vez elaborado el diseño en el entorno gráfico, seleccione la opción para añadir las funciones que se desea que realice el robot. Los periféricos y sus situaciones se cargarán dinámicamente dependiendo de los periféricos presentes en el diseño elaborado.

4.1.3.3 Acciones.txt

Este fichero almacenará las posibles acciones que podrá realizar el robot en una de las situaciones definidas en el fichero anterior. Inicialmente contará con un par de acciones, con su correspondiente función para algunos de los periféricos definidos en el ***PFC [6]***. Estas acciones podrán ser ampliadas desde el nivel de dificultad Avanzado, en la cual, el usuario podrán añadir o eliminar las acciones según su necesidad. También cuenta con la funcionalidad para poder dar de alta o de baja las acciones correspondientes a un periférico. Cada acción también tendrá ligada a una función que ha sido previamente definida en la API o librería C elaborada para ello.

La estructura que tendrá este fichero es la siguiente:

- La primera línea del fichero corresponderá a una cabecera, en la cual se indicará:

  - **Periférico:** definirá el nombre del sensor o del actuador.
  - **Acción:** acción que va a llevar a cabo el periférico.
  - **Función:** es la función que se ha añadido a la API o librería y la cual corresponde a la acción que se va a dar de alta en la aplicación.

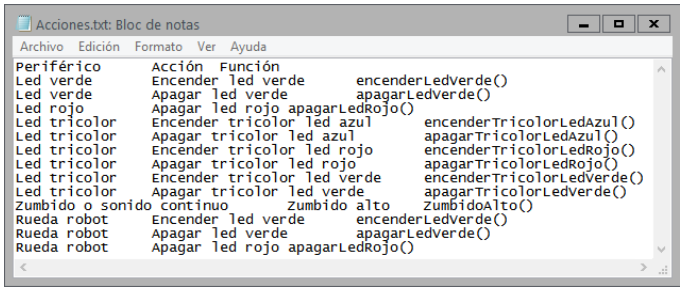

Ilustración 14: Fichero Acciones.txt

- Las líneas siguientes estarán compuestas por los distintos periféricos con las acciones que ya estuviesen definidas o que hayan sido dadas de alta por el usuario.

Cuando el usuario seleccione una situación en la interfaz gráfica, nuestra aplicación mostrará una ventana en la que cargará de forma dinámica las posibles acciones que podrá llevar el robot en tal situación.

4.1.4 Funciones de acceso a los Datos *(Controlador)*

Tras el análisis realizado a lo largo de este apartado, se ha visto que la parte asociada al controlador será el conjunto de funciones o métodos que permitirá a la vista interactuar con el modelo de datos y a las peticiones que realiza el usuario en la vista. El conjunto de funciones se han definido teniendo en cuenta el único subsistema que se ha descrito en la fase de análisis que es el Usuario.

En el Anexo C se detalla cada una de las funciones correspondientes al subsistema Usuario, indicando para cada una su descripción, los parámetros de entrada, la salida esperada, los requisitos cubiertos por cada función. En la tabla 1 se expone el conjunto de funciones resumiendo los detalles más relevantes.

Función	Parámetros de entrada	Salida	Requisito
checkSerialPort	-	Valor entero	RF1-U1
readPeripheralsInRobot	-	Valor lógico	RF1-U1, RF2-U2
setNumMaxPeripherals	-	Valor lógico	RF2-U2, RF3-U3
viewDetailsPeripheral	-	Valor lógico	RF2-U2, RF4-U4
openTestPeripheralsFrame	-	Valor lógico	RF1-U1, RF5 -U5
addPeripheralWorkspace	-	Valor lógico	RF1-U1, RF6 –U6
deletePeripheralWorkspace	Peripheral	Valor lógico	RF7–U7
deleteAllPeripheralsWorkspace	-	Valor lógico	RF7–U7
readSituationOfRobot	-	Valor lógico	RF8-U8
addListenerSituations	Situación	-	RF9-U9
readActionsRobot	-	Valor lógico	RF10–U10
viewActionsRobot	mapPeripheralAction	Valor lógico	RF11–U11
updatePanelActionsRobot	-	Valor lógico	RF12–U12
createProgram	-	Valor lógico	RF13–U13
loadHEXToMicro	-	Valor entero	RF14–U14
doSaveAPEFile	-	Valor lógico	RF15–U15
doSave**As**APEFile	-	Valor lógico	RF15–U15
doOpenAPEFile	-	Valor lógico	RF16–U16
registerPeripheralInFile	perif, numMaximo	Valor lógico	RF17–U17
deregisterPeripheralInFile	Peripheral	Valor lógico	RF18–U18
registerSituationInFile	peripheral, i_d, situation, function	Valor lógico	RF19–U19
deregisterSituationsInFile	Situation	Valor lógico	RF20–U20
registerActionInFile	peripheral, action, function	Valor lógico	RF21–U21
deregisterActionInFile	Action	Valor lógico	RF22–U22

Tabla 1: Funciones de la API

Tras haber definido las funciones o métodos se puede observar que hay funciones que no interactúan directamente con el modelo de datos. Estas funciones son: *addPeripheralWorkspace, deletePeripheralWorkspace, deleteAllPeripheralsWorkspace, doSaveAPEFile, doSaveAsAPEFile, doOpenAPEFile* que aunque no interactúen directamente con el modelo de datos sí que hacen uso de los datos almacenados en el modelo.

4.1.5 Prototipo de la Interfaz Gráfica de Usuario *(Vista)*

En este apartado se mostrará una primera versión de la interfaz gráfica de la apariencia final de la aplicación anterior a la fase de desarrollo.

En primer lugar se definirá la estructura de cada ventana de la interfaz gráfica, a las cuales, se les ha asociado un nombre para poder referirnos a ellas en el mapa de navegación que se elaborará más adelante. En el Anexo D se indica el diseño conceptual de las ventanas de la aplicación.

4.1.5.1 Esquema de navegación entre ventanas

En este apartado se muestra el esquema de navegación del usuario entre las diferentes ventanas presentes en la interfaz gráfica. En la Ilustración 15 se ve todas estas ventanas en miniatura que tendrán asociada el nombre de las ventanas definidas en el Anexo D. También se muestra las acciones que el usuario ha llevado a cabo para provocar el cambio de una ventana a otra.

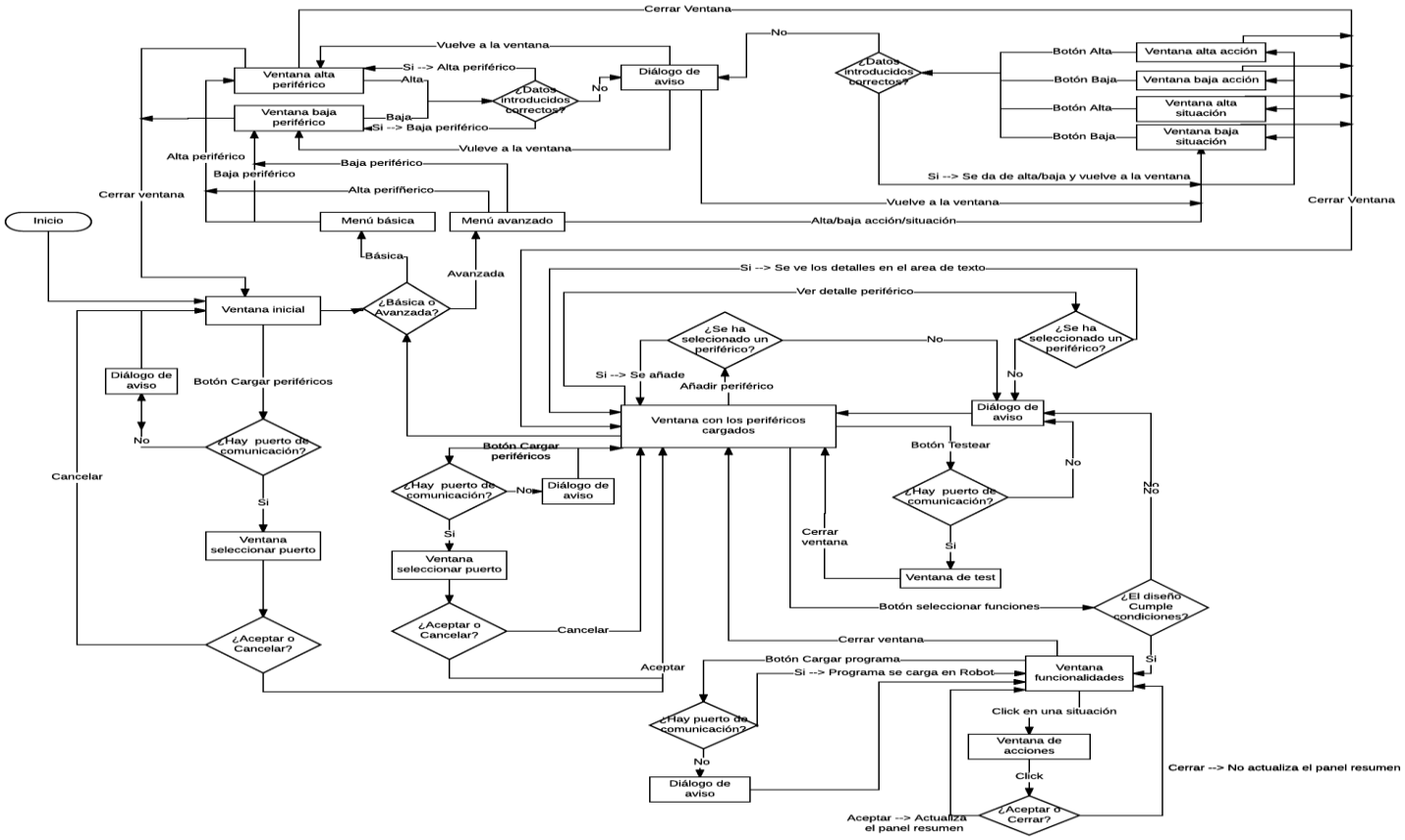

Ilustración 15: Diagrama de navegación entre las ventanas de la interfaz gráfica

A continuación se va a describir de forma general el diagrama de la Ilustración 15 sin hacer hincapié en los controles de errores ya que en el Anexo H se describirá en más detalle cada una de las ventanas de la aplicación, sus errores y la causa de los mismos.

Desde la ventana inicial el usuario solo podrá cargar los periféricos en la interfaz gráfica y dar de alta o de baja periférico, acciones y situaciones del robot según el modo de dificultad. Cuando el usuario pulse sobre el botón "Cargar periféricos" el sistema muestra la ventana para seleccionar el puerto de comunicación y posteriormente cargar los periféricos en los distintos componentes de la interfaz gráfica. Una vez cargado los periféricos el usuario podrá ir añadiéndolos al espacio de trabajo. Cuando tenga elaborado el diseño en el entorno de trabajo, el usuario podrá seleccionar las funciones que desea que realice el robot, para ello tiene que pulsar sobre el botón "Seleccionar acciones". El sistema mostrará la pantalla "Ventana funcionalidades" en la cual se muestran las posibles situaciones en las que se puede encontrar el robot. Cada vez que el usuario seleccione una de estas situaciones el sistema mostrará al usuario la pantalla "Ventana acciones" la cual permitirá seleccionar las acciones que cada periférico del robot va a realizar en la situación indicada. Una vez se haya definido todas las funcionalidades deseadas, el usuario podrá programar el robot con todo lo que ha definido, para ello tendrá que pulsar sobre el botón "Cargar programa" que posteriormente cargará el programa en el robot.

Respecto a las operaciones de alta y baja de periféricos, situaciones y acciones del robot ya se ha explicado en apartados anteriores su funcionamiento. De todas formas en el Anexo H se explica su funcionamiento de forma más detallada.

4.1.6 Protocolo de comunicación entre el pc y el robot

El robot elaborado en el ***PFC [6]*** dispondrá de un conector tipo MiniUSB, Desde el primer momento el protocolo de comunicación que se cree más conveniente es el protocolo USB. Este protocolo de comunicación tiene muy buenas características y una disponibilidad inmediata, puesto que al ser por cable no realiza ningún tipo de interferencia con las señales de radiofrecuencia como puede suceder con el protocolo de comunicación Bluetooth.

La conexión de los dispositivos USB está disponible en la mayoría de los equipos, y por tanto aumenta la compatibilidad.

En internet existen gran variedad de librerías que nos permita conseguir el control USB de forma rápida y sencilla, en este TFG se utiliza en concreto la librería LUFA, que permite implementar este protocolo USB en el microcontrolador.

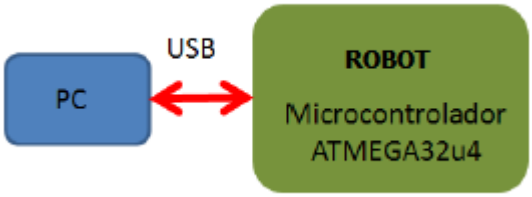

Ilustración 16: Esquema final del sistema completo

5 Desarrollo

En este apartado se traducirá el diseño realizado en el capítulo anterior a código de un lenguaje de programación, en concreto JAVA. En este punto se recogen las características más relevantes del proceso de desarrollo de nuestra aplicación. Para ello se describe las tecnologías utilizadas, el entorno de desarrollo para su implementación, el material usado y los módulos creados siguiendo las pautas marcadas en la fase de análisis y diseño de la aplicación. Finalmente se terminará detallando la interfaz final de la aplicación diseñada a partir del prototipo presentado en la fase de diseño.

5.1 Tecnologías utilizadas

Desde un principio se ha mencionado que la aplicación deberá funcionar independientemente del sistema operativo y del ordenar que se utilice. Por lo tanto, el lenguaje de programación adecuando para el desarrollo de esta aplicación es JAVA, ya que los programas elaborados en este lenguaje podrán ser ejecutados en cualquier plataforma y sobre todo ofrece la portabilidad de la aplicación.

5.2 Entorno de desarrollo

Para el desarrollo de la aplicación se propuso usar el lenguaje de programación JAVA. JAVA es un lenguaje de programación orientado a objetos [21] desarrollado por Sun Microsystems. Java, se trata de un lenguaje de alto nivel, muy aproximado al lenguaje natural lo cual hace más fácil la comprensión del código y su implementación.

La portabilidad de los programas generado con el lenguaje de programación java es un punto a favor para este tipo de lenguaje, debido a que el código que se realiza, se traduce a un código intermedio independiente de la máquina en la que se va a usar. Por lo que gracias a estas características los programas implementados en este lenguaje podrán ejecutarse en distintas plataformas. Por lo tanto, el desarrollo de este proyecto se va a realizar con la herramienta NetBeans [20] ya que permite la facilidad de crear aplicaciones de escritorio por medio de la biblioteca gráfica Swing.

5.3 Estructura de paquetes

Para seguir el patrón de diseño MVC hay que estructurar las clases de una manera ordenada. Para ello se han creado los siguientes paquetes.

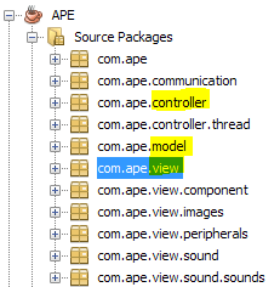

Ilustración 17: Estructura de paquetes del proyecto

Como se puede observar en la Ilustración 17 hay paquetes relacionados con la implementación del patrón de diseño MVC. El modelo tendrá una única clase que implementará toda la lógica de la aplicación. También se puede observar el paquete que contiene todos los controladores que mapean las acciones realizadas por el usuario desde la interfaz gráfica. Por último, está el paquete que contienen todas las vistas con las que interactuará el usuario.

5.4 Desarrollo de módulos

En este apartado se detalla la implementación de cada uno de los módulos centrándonos en las distintas funciones, para de esta forma poder ver como se realiza cada uno de las acciones del sistema.

5.4.1 Modelo

El modelo estará en el paquete ***com.ape.model*** donde existirá una única clase con el nombre Context.java. Esta clase tendrá implementado la parte lógica del programa, es decir, la parte que da resultado y la que resuelve el problema.

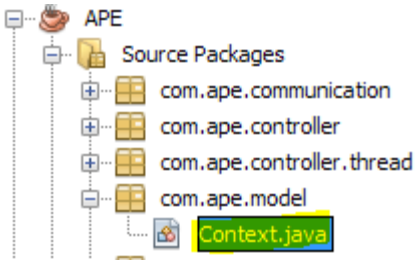

Ilustración 18: Paquete modelo

En este trabajo el modelo será el responsable de acceder a los datos almacenados en los ficheros de texto que se han descrito en la fase de diseño.

Las funciones de acceso a estos ficheros se ponen a continuación:

- ***readFileListPeripherals():*** esta función se encarga de leer los periféricos del fichero Periféricos.txt.
- ***readFileListActions():*** esta función se encarga de leer las acciones del fichero Acciones.txt.
- ***readFileListSituations():*** esta función se encarga de leer la situaciones del fichero Situaciones.txt.

Por otro lado, el modelo va a definir las funcionalidades que deberá tener la aplicación para cumplir con los requisitos especificados en la fase de Análisis (ver Anexo B).

5.4.2 Controlador

El controlador será el responsable de recibir los eventos de entrada que el usuario lleva a cabo sobre la interfaz gráfica ya sea pulsando sobre un botón, un cambio en un campo de texto, etc. Generalmente contendrá las reglas de gestión de estos eventos. Estas acciones suponen peticiones al modelo o a las vistas.

Tal y como se ha mencionado en apartados anteriores, las peticiones que se realizarán al modelo son básicamente para consultar o modificar los fichero Perifericos.txt, Situaciones.txt y Acciones.txt y para actualizar los datos en la vista, un área de texto, una lista de objetos, etc.

El controlador estará en el paquete ***com.ape.controller*** que tendrá implementado varias clases correspondientes a las peticiones que el usuario puede llevar a cabo en la interfaz gráfica.

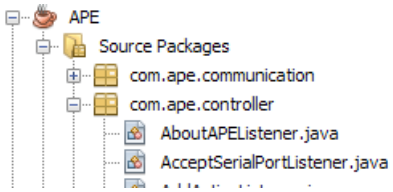

Ilustración 19: Paquete controlador

Todas estas clases implementarán la interfaz ***ActionListener*** y tendrán el formato de la Ilustración 53 el Anexo E.

En cada una de las clases que define este paquete, se sobrescribirá el método ***actionPerformed*** definiendo las acciones que la aplicación tendrá que realizar cuando el sistema capture el evento. En este ejemplo, el evento que se ejecuta se añade al botón "Ver detalles" tal y como se indica en la Ilustración 54 del Anexo E.

5.4.3 Vista

Para finalizar el desarrollo falta implementar las vistas que dispondrá la interfaz gráfica. Se crearán vistas para cada una de las ventanas que se mostrarán al usuario.

Las vistas están implementadas en el paquete ***com.ape.view*** que hace uso de los paquetes ***com.ape.view.component, com.ape.view.images, com.ape.view.peripherals, com.ape.view.sound y com.ape.view.sound.sounds.*** Más a delante se explicará la funcionalidad de cada uno de estos paquetes.

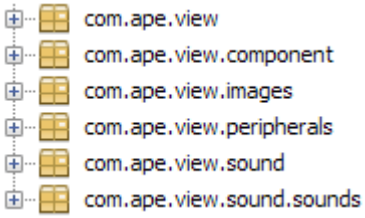

Ilustración 20: Paquete Vista

A continuación se define de forma clara y concisa cada una de las clases correspondientes al paquete ***com.ape.view:***

- ***AboutAPEFrame:*** muestra la información, versión y nombre de la aplicación.
- ***ActionsFrame:*** muestra las acciones que dispone cada periférico del robot.
- ***AddActionFrame:*** muestra la ventana en la que el usuario podrá dar de alta un periférico y su acción.
- ***AddPeripheralFrame:*** ventana en la que el usuario podrá dar de alta un periférico.
- ***AddSituationFrame:*** en esta ventana el usuario podrá dar de alta un periférico y la situación correspondiente al mismo.
- ***DeleteActionFrame:*** en esta ventana el usuario podrá dar de baja la acción que desee.
- ***DeletePeripheralFrame:*** aquí el usuario podrá dar de baja el periférico que desee.
- ***DeleteSituationFrame:*** en esta ventana se podrá dar de baja la situación de un periférico.
- ***OpenAPEFrame:*** esta es la ventana principal que incorpora todas las funciones que el usuario puede llevar a cabo, desde cargar periféricos a dar de alta, baja periféricos.
- ***SerialPortFrame:*** ventana en la que el usuario seleccionará el puerto de comunicación.
- ***SelectActionRobotFrame:*** muestra todas las funcionalidades que el usuario podrá definir en la interfaz gráfica.
- ***TestPeripheralsFrame:*** es la ventana en la que se realizará el test de cada uno de los periféricos disponibles en la aplicación.

Todas estas clases extenderán de la clase JFrame que se encuentra en el paquete ***javax.swing***.

A continuación se pasa a describir el resto de paquetes que serán utilizados por las vistas descritas anteriormente.

- ***com.ape.view.component:*** este paquete contiene dos clases: JTextFieldHint y TransparentPanel.

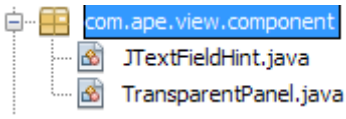

Ilustración 21: Paquete com.ape.view.component

La clase ***JTextFieldHint*** es usada para mostrar un aviso al usuario en aquellos campos donde se deberá introducir texto.

Ilustración 22: Uso de la clase JTextFieldHint

La clase ***TransparentPanel*** es usada para poder pintar el logo de la aplicación en la interfaz gráfica.

- ***com.ape.view.images:*** este paquete contiene todas las imágenes correspondientes a cada uno de los periféricos.

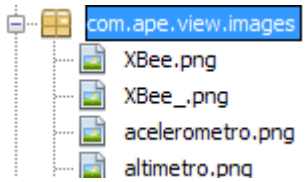

Ilustración 23: Paquete com.ape.view.images

- ***com.ape.view.peripherals:*** en este paquete se implementan las clases correspondientes a cada uno de los periféricos que se han definido en el ***PFC [6].***
  - ***Clase Peripheral:*** esta clase extiende de JLabel e implementa la interfaces ***MouseListener, MouseMotionListener, Serializable.***
  - ***Clase Abstracta Sensor:*** que extiende de la clase ***Peripheral.***
  - ***Clase Abstracta Actuator:*** que extiende también de la clase ***Peripheral.***

Se ha seguido esta estructura para tener modulado el código y poder distinguir los periféricos de tipo sensor y actuador. Los periféricos definidos como sensores extenderán de la clase abstracta ***Sensor*** mientras que los periféricos que sean del tipo actuador extenderán de la clase abstracta ***Actuator.***

La función abstracta que dispone la clase Sensor es:

```
public abstract void putSensorInRobot();
```

La función abstracta que dispone la clase Actuator es:

```
public abstract void putActuatorInRobot();
```

Ambas funciones tienen la finalidad de posicionar el periférico en el diseño elaborado en el entorno de trabajo. Cada uno de los periféricos tendrá definida su posición en el entorno gráfico, de esta forma cada vez que el usuario arrastre el periférico a su posición correspondiente el sistema lo colocará automáticamente sobre ella.

La clase Peripheral tiene la función ***setLocationPeripheral*** que será sobrescrita por todos los periféricos. Su finalidad será posicionar el periférico cuando se redimensione la ventana principal. De esta forma se puede conservar la posición del periférico en el diseño expuesto en el entorno de trabajo.

- ***com.ape.view.sound:*** la aplicación emitirá sonidos cuando un periférico:
  - Es arrastrado.
  - Es colocado en el diseño en su posición correspondiente.
  - Es movido a la papelera de reciclaje.

- Es eliminado.
- Es pulsado.

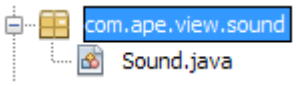

Ilustración 24: Paquete com.ape.view.sound

Este paquete únicamente tendrá la clase ***Sound.java*** (Ver Ilustración 55 de Anexo E) que en ella se implementara la funcionalidad correspondiente para gestionar el sonido que se deberá de emitir cuando el usuario realice alguna de las acciones indicadas anteriormente.

- ***com.ape.view.sound.sounds:*** este paquete contiene los sonidos que se usarán en la aplicación.

- ***com.ape.main:*** este paquete contiene la clase Main.java que tiene el método main que ejecuta la aplicación. Éste método hace una llamada a la función ***startAPE*** que llama al método ***startAPEFrame*** que será la que finalmente inicie a ventana principal de la aplicación**.**

Nos queda detallar el paquete ***com.ape.communication*** que se explicará en el apartado siguiente ya que implementa las funciones utilizadas para el protocolo de comunicación USB.

5.5 Protocolo de comunicación y librerías utilizados

En este apartado se estudiarán los protocolos de comunicación USB utilizados tanto en la parte de la aplicación como en la parte del robot. En ambos casos se utilizarán diferentes librerías para establecer comunicación por el puerto USB.

Veremos cómo se realiza el intercambio de información entre el pc y el robot, sobre todo cuando el usuario carga los periféricos en la interfaz gráfica, cuando realiza el test de cada periférico y cuando carga todas las funcionalidades al robot.

La aplicación dispondrá de una librería implementada en lenguaje de programación C la cual contiene todas las funciones que la aplicación utilizará para traducir toda la funcionalidad, que se ha seleccionado previamente en la aplicación, a un programa en C el cual será compilado con las herramientas correspondientes para obtener el programa hexadecimal que será el que finalmente se cargue en el robot.

Finalmente se expondrá en una tabla todos los comandos utilizados para el intercambio de información entre el ordenador y el robot.

5.5.1 USB DFU Bootloader [22]

Tal y como se ha comentado en la fase de diseño, se va a usar el dispositivo MiniUSB que dispondrá el robot elaborado en el ***PFC [6]*** para programar cualquier tipo de

funcionalidad, por lo que será necesario programar previamente en el microcontrolador del robot un gestor de arranque.

Los microcontroladores con dispositivos de interfaz USB están configurados con un gestor de arranque USB (USB Bootloader) que se encuentra en la sección de arranque de la memoria Flash en el chip del microcontrolador.

El gestor de arranque gestiona el protocolo de comunicación USB y realiza operaciones de lectura / escritura a la memoria en el chip (Flash / EEPROM).

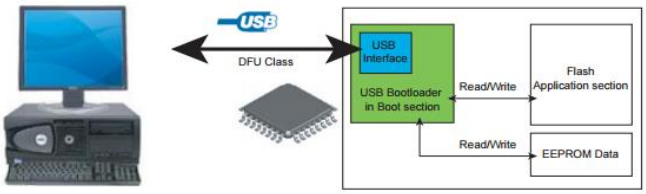

Ilustración 25: Entorno físico

5.5.1.1 Proceso de arranque

El gestor de arranque se ejecuta en cada secuencia de reset/encendido y primero lleva a cabo el proceso de arranque para saber si debe iniciar el USB DFU o la aplicación.

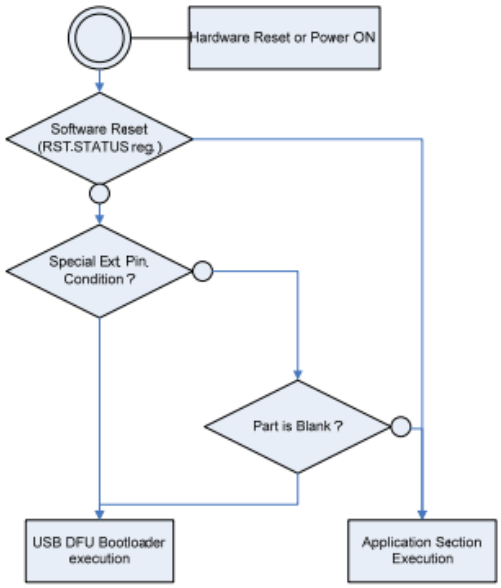

Ilustración 26: Proceso de arranque

5.5.1.2 Cargar USB DFU Bootloader en el microcontrolador del robot

En este proyecto se hace uso de este gestor de arranque, con lo cual se tendrá que instalar en el microcontrolador del robot. Para ello será necesario bajarse el programa hexadecimal ATMega32U4-usbdevice_dfu-1_0_0.hex proporcionado por Atmel [23].

Una vez se disponga del .hex, hay que programarlo al microcontrolador del robot a través de la herramienta Atmel Studio 6.0 [24] utilizando el Programador AVRISP mkII que es necesario para cargar cualquier programa en el procesador.

Ilustración 27: Programador AVRISP mkII

En el Anexo F se indican los pasos a seguir para cargar el USB DFU bootloader a través del Programador AVRISP mkII y la instalación del driver para Windows.

Tras haber programado el bootloader siguiendo los pasos del Anexo F, el robot ya está listo para cargar los programas en el microcontrolador, ya sea por la herramienta FLIP o por línea de comandos.

En este trabajo no se utilizará la herramienta Flip, ya que en este trabajo se tendrá que cargar los programas desde la interfaz gráfica. Por lo tanto, se utilizarán los comandos correspondientes para cargar los programas en el microcontrolador a través del puerto USB. El comando usado es:

batchisp -device atmega32u4 -hardware usb -operation erase f memory flash blankcheck loadbuffer program.hex program verify start reset 0

5.5.2 Protocolo de comunicación en Robot

Como ya se ha mencionado anteriormente el protocolo de comunicación entre el pc y el robot, se hace por medio del USB, aprovechando el conector MiniUSB disponible en el robot elaborado en el ***PFC [6]***.

Para utilizar este protocolo de comunicación con Atmel y AVR es necesario utilizar una librería especial. Hay gran variedad de librerías que permiten implementar el protocolo de comunicación USB.

En este TFG se hace uso de la librería LUFA, esta librería tiene la peculiaridad de hacer que los microcontroladores sean compatibles con el protocolo de comunicación USB. Nosotros utilizaremos la Demo VirtualSerial, ya que la comunicación se realiza a través del envío de datos por un puerto serie, siendo necesario realizar algunas modificaciones en el código.

5.5.2.1 LUFA

La comunicación USB por un puerto serie se realiza a través de un Endpoint o buffer. Cuando el pc/host va a enviar datos al dispositivo lo escribe en un Endopoint, y el dispositivo leerá esta información cuando lo desee. Un Endpoint es un canal unidireccional, puede ser IN que va desde el dispositivo al host, OUT que va desde el Host al dispositivo o IN/OUT bidireccional, que la comunicación se realiza en ambos sentidos.

5.5.2.2 Librería Demo VirtualSerial de LUFA

Antes de poder programar el micro con la librería LUFA, concretamente la demo VirtualSerial, se tendrá que instalar los drivers necesarios para compilar y obtener el fichero VirtualSerial.hex que es el programa que se cargará en el microcontrolador.

Antes de poder programar el micro con la librería LUFA, es necesaria la instalación de los drivers del AVRISP mkII:

- ***Avr-gcc:*** es el compilador de AVR.
- ***Avrdude:*** permiten utilizar las memorias Flash y EEprom de
- ***Avr-libc:*** librería para la programación C de los microcontroladores con AVR.

La demo VirtualSerial tiene el contenido de la Ilustración 28. A continuación se describirá de forma inmediata aquellos ficheros más relevantes.

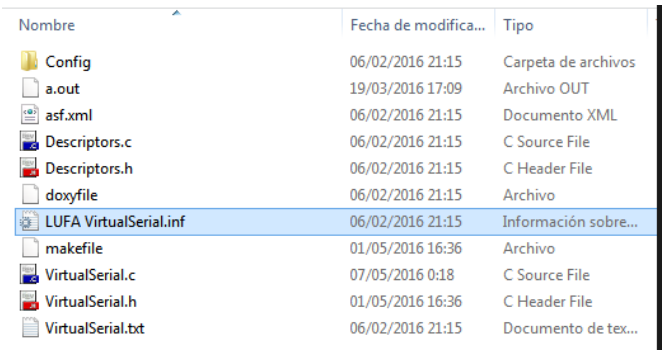

Ilustración 28: Fichero de la Demo VirtualSerial

A través del fichero Descriptors.c y Descriptors.h el equipo tiene información acerca del AVR que utiliza, sus características y los endpoints disponibles. La parte más importante a tener en cuenta son los códigos VID y PID que se encuentran en el fichero Descriptors.c y corresponden con el número del proveedor USB y del producto.

```
.VendorID                 = 0x03EB,
.ProductID                = 0x2044,
```

Ilustración 29: VID y PID

En este proyecto el fichero que se ha modificado para acoplarlo a nuestra necesidad es el archivo VirtualSerial.c, ya que es el archivo donde se encuentra el programa principal (main) y donde se tendrá que definir como se reciben y se envían los datos a través del protocolo de comunicación USB. La función que se utiliza para la recepción de los datos

es: ***CDC_Device_ReceiveByte*** que retorna el byte enviado desde el pc y en caso de que no se haya enviado nada se retornará un valor negativo. En la Ilustración 56 del Anexo E se adjunta el código utilizado para la recepción y el envío de datos.

En la Ilustración 56 del Anexo E se observa que cuando se reciben bytes desde el ordenador se va almacenando en un buffer y tras dejar de recibir bytes se comprueba que el dato recibido tenga una longitud 2, que corresponde a la longitud de los comandos que se definirán más adelante. Una vez se ha recibido un comando, se llama a la función ***procesarComando*** que se encargará de procesar el comando y llevar acabo las acciones correspondientes a dicho comando. En esta función se incorpora toda la funcionalidad para realizar el testeo de cada uno de los periféricos que dependiendo del comando recibido, por ejemplo, moverá una rueda, encenderá un led, etc.

Todo esto ficheros vienen acompañado de un Makefile (ver Ilustración 30) que será necesario modificarlo ya que se debe definir el nombre del micro utilizado (MCU), así como el AVR, la frecuencia CPU (F_CPU) y la frecuencia USB (F_USB).

```
12  # Run "make help" for target help.
13
14  MCU          = atmega32u4
15  ARCH         = AVR8
16  BOARD        = USBKEY
17  F_CPU        = 8000000
18  F_USB        = $(F_CPU)
19  OPTIMIZATION = s
```

Ilustración 30: Makefile LUFA

Una vez se haya definido en el fichero VirtualSerial.c las acciones que tendrá que realizar el robot para cada uno de los comandos recibidos, se procederá a compilar la demo con el Makefile para obtener el programa hexadecimal (VirtualSerial.hex) que es el que finalmente se cargará en el micro del robot. Este programa se cargará en el micro cada vez que el usuario seleccione el botón "Cargar periféricos" y "Testear periféricos" de la interfaz gráfica. Esto se explicará más detalladamente en apartados siguientes.

5.5.2.3 LUFA CDC-ACM Virtual Serial Port

Una vez se haya cargado el programa VirtualSerial.hex en el micro del robot, cuando se conecte el USB al micro inmediatamente el sistema operativo reconocerá el dispositivo tal y como se puede observar en el Administrador de dispositivos (ver Ilustración 72 Anexo G).

En el Anexo G, se explican los pasos que hay que llevar a cabo para instalar el controlador para el dispositivo LUFA CDC Demo.

Tras la instalación del controlador LUFA CDC, el microcontrolador está listo para recibir información procedente del ordenador y enviar datos al pc a través del puerto USB.

Aquí se finaliza la implementación e instalación del protocolo de comunicación en la parte del microcontrolador. En el apartado siguiente vamos a centrarnos en el protocolo de comunicación que usa la interfaz gráfica para enviar y recibir datos a través del puerto USB.

5.5.3 Protocolo de comunicación Interfaz gráfica

En este apartado se indicará el protocolo de comunicación USB utilizado en la interfaz gráfica para el envío y recepción de datos a través del puerto serie.

Para utilizar este protocolo de comunicación en JAVA es necesario utilizar una librería especial. Hay muchas librerías que permiten implementar el protocolo de comunicación USB entre ellas esta JSSC.

5.5.3.1 Librería utilizada: JSSC (java-simple-serial-connector)

En este TFG se hace uso de la librería JSSC (java-simple-serial-connector) [25]. Es una librería para poder trabajar con el puerto serie desde java. Una de las posibilidades que ofrece la librería JSSC es poder obtener los nombres de los puertos, leer y escribir datos, líneas de control RTS y DTR, etc.

En este proyecto se ha definido un paquete ***com.ape.communication*** con todas las funciones necesarias para la recepción y envío de datos a través del puerto serie. También se ha añadido la librería JSSC.jar al proyecto para poder hacer uso de todas estas funciones.

Este paquete contendrá la clase ***CommunicationProtocol.java*** que tendrá implementada las funciones necesarias para el intercambio de información entre el robot y el pc.

La interfaz gráfica hará uso de estas funciones cada vez que el usuario tenga la necesidad de cargar y de testear los periféricos disponibles en el robot. En este último es cuando más se van a utilizar estas funciones, ya que se deberán enviar y recibir datos al y del robot para comprobar el funcionamiento de cada uno de los periféricos.

5.5.4 Intercambio de información entre el pc-robot

Se ha descrito en apartados anteriores que el micro del robot tendrá cargado un gestor de arranque (DFU bootloader) en la memoria flash, que cada vez que se ejecute la secuencia reset/encendido lleva a cabo el proceso de arranque para saber si debe iniciar el USB DFU o la aplicación.

También se ha indicado que se utilizará la demo VirtualSerial de la librería LUFA para implementar el protocolo de comunicación. Tal y como se ha dicho anteriormente, en el fichero VirtualSerial.c se definirán las acciones que realizará el robot para cada uno de los comandos recibidos. Tras definir estas acciones se procederá a compilar la demo con el Makefile para obtener el programa hexadecimal (VirtualSerial.hex) que es el que finalmente se cargará en el micro del robot.

5.5.4.1 Cargar periféricos

Cuando el usuario inicia la aplicación tendrá que cargar los periféricos en la interfaz gráfica, para ello deberá pulsar sobre el botón “Cargar periféricos”. Tras esta acción el sistema internamente realizará es lo siguiente:

- Obtendrá todos los puertos de comunicación conectados al pc.

- Comprobará si el micro ya tiene cargado el programa VirtualSerial.hex. Para ello el sistema enviará el comando "-C" a cada uno de estos puertos y si no recibe respuesta el sistema descarta dicho puerto. Esta comprobación lo realizará la función ***testPortComunication*** (ver Ilustración 57 Anexo E).
- Si el puerto al que se le ha enviado el comando responde con la cadena "Conectado", inmediatamente el sistema almacenará este puerto para posteriormente mostrarlo en la ventana donde se seleccionará el puerto de comunicación.
- Si el sistema no recibe ninguna respuesta, procederá a cargar el programa hexadecimal VirtualSerial.hex. Esto lo llevará acabo la función ***programMicroProtComunication()*** (ver Ilustración 58 Anexo E), que básicamente lo que realizará será cargar en el micro el programa hexadecimal VirtualSerial.hex ejecutando el siguiente comando:

  bin/batchisp.exe -device atmega32u4 -hardware usb -operation erase f memory flash blankcheck loadbuffer bin/virtualserial.hex program verify start reset 0

En la Ilustración 59 del Anexo E se puede observar todo el proceso que se ha indicado anteriormente.

Por lo tanto, si el micro ya tiene cargado el programa VirtualSerial.hex el sistema no lo cargará de nuevo, pero si no lo tiene el sistema lo cargará automáticamente ejecutando la función ***programMicroProtComunication().***

5.5.4.2 Testear periféricos

Como ya se ha mencionado en varias ocasiones, el sistema tendrá la posibilidad de testear cada uno de los periféricos disponibles en el robot pulsando sobre el botón "Test periféricos". Tras esta acción el sistema internamente realizará las mismas comprobaciones que se han indicado en el apartado ***Cargar periféricos***.

Es necesario que antes de acceder a esta ventana el sistema tenga la certeza de que el programa hexadecimal VirtualSerial.hex está cargado en el micro. Esto es importante ya que es la parte en donde más intercambio de datos se realizará entre el pc y el robot, debido a que la aplicación enviará un comando u otro a través del puerto de comunicación para comprobar el funcionamiento del periférico deseado. Cada periférico tendrá asociado su comando, con lo cual, en el fichero VirtualSerial.c se ha definido previamente el comando correspondiente a cada periférico y la accione que tiene que realizar.

5.5.4.3 Cargar programa a robot

En la aplicación existe la posibilidad de cargar al robot todas las funcionalidades previamente definidas por el usuario. Cuando el usuario pulsa sobre el botón "Cargar funciones al robot", el sistema internamente realizará lo siguiente:

- Obtendrá los puertos de comunicación conectados al pc.
- Comprobará cada puerto de comunicación con la función ***testPortComunication***.

- Si no hay ningún puerto de comunicación que corresponda al robot, el sistema creará el programa principal (main.c) con todas las funcionalidades que ha definido el usuario. Posteriormente generara el programa hexadecimal y será cargado en el micro de robot.
- Si hay algún puerto de comunicación que corresponda al robot, el sistema informará al usuario que presione el botón reset del robot para entrar en modo programación. Esto es necesario ya que se va a cargar el programa hexadecimal correspondiente al programa main.c que contiene toda las funcionalidades que ha definido el usuario en la interfaz.

En la Ilustración 60 del Anexo E hace referencia al fragmento de código que realiza lo comentado anteriormente.

5.5.5 Librería C con todas las funciones

Cuando el usuario pulsa sobre el botón "Cargar funciones del robot", el sistema generará automáticamente un programa **main.c** traduciendo todas las funcionalidades que ha definido el usuario a lenguaje de programación C. Para ello hará uso de las librerías: Rutinas.h (ver Ilustración 61 del Anexo E) y Rutinas.c (ver Ilustración 62 del Anexo E) donde estarán definidas todas estas funcionalidades.

No se ha puesto el código de todas las funciones que se ha implementado en estos ficheros ya que la memoria sería excesiva. Hay que decir que esta librería no incorpora todas las funciones que debería ya que esta parte era labor del ***PFC*** [6] pero dado a que aún no se ha podido disponer de una versión definitiva de sus resultados, para la presentación de este TFG se ha implementado, por mi parte, varias funciones que serán usadas para comprobar el correcto funcionamiento de la aplicación en cuanto a la operación de cargar las funciones al robot.

5.5.6 Comandos utilizados

En este apartado se definirán los comandos que serán enviados y recibidos a través del puerto de comunicación, tanto por la interfaz gráfica como por el robot (ver tabla 2 del Anexo I).

5.6 Interfaz de Usuario

5.6.1 Introducción

La realización de la interfaz gráfica ha sido la tarea que nos ha llevado más tiempo ya que era importante su realización para que el usuario entendiera bien la interfaz para poderse manejar con facilidad, incluso sin leer este apartado. En todo momento intentamos que la interfaz fuera lo más sencilla, clara y que nos mostrara los errores en mensajes de aviso en el caso de no poderse realizar alguna acción en el sistema.

En el Anexo H mostraremos las interfaces principales de nuestra aplicación y describiremos las acciones que se llevarán a cabo en cada una de ellas así como los controles de errores y avisos que se mostrarán al usuario cada vez que haya algún tipo de error en el sistema.

6 Integración, pruebas y resultados

Para probar finalmente que el sistema desarrollado en este proyecto funciona correctamente, en primer lugar se va a comprobar que la comunicación entre la aplicación y el robot es satisfactoria.

Como ya se comentó al principio del documento este proyecto iba emparejado con el ***PFC [6]***, pero dado a que aún no se ha podido disponer de una versión definitiva de sus resultados se ha utilizado la placa diseñada por otro ***PFC [26],*** de características básicas similares**.** Esta placa también servirá de ayuda para las pruebas del sistema completo que se realizarán posteriormente.

Se trata de una placa básica en la que se facilita el acceso a los pines del microcontrolador, y que cuenta con utilidades como mini-USB, alimentación, 3 pulsadores, dos LEDs y acceso ISP (ver Ilustración 31).

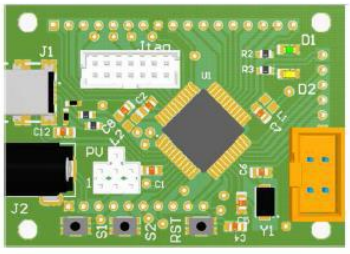

Ilustración 31: Placa utilizada para las pruebas del sistema

6.1 Comunicación

La comunicación entre el ordenador y el robot es fundamental para que el sistema completo funcione correctamente y el intercambio de información entre ambas partes se realice de forma satisfactoria. Tal y como se ha hablado en la fase de desarrollo, esta comunicación se realizará usando el puerto MiniUSB presente en la placa mencionada anteriormente.
En primer lugar se deberá cargador el programa Bootloader DFU (ver Anexo F) en la placa e instalar el controlador LUFA CDC en el ordenador (ver Anexo G). Una vez realizado esto, se procederá a conectar la placa al ordenador, donde éste inmediatamente reconocerá el dispositivo LUFA-CDC-ACM Virtual Serial Port (COM8). La placa inicialmente tendrá cargado el programa VirtualSerial.hex que permite al ordenador reconocer que está conectado este dispositivo.
Cuando se conecte la placa al ordenador por el puerto USB se podrá ver en la Ilustración 32 que la interfaz gráfica lo detecta.

Ilustración 32: Reconocimiento del puerto de comunicación

A parte de aquí el usuario puede interactuar con la aplicación según sus necesidades.

6.2 Pruebas

En este apartado se explicarán las pruebas seguidas para comprobar el correcto funcionamiento de los módulos, la integración con el sistema, las pruebas del sistema completo.

A continuación se muestra la planificación de las pruebas que se va a realizar sobre el sistema:

- ***Pruebas unitarias o de caja blanca:*** pruebas que se realizan sobre las funciones internas de un módulo. De esta forma se comprueba la lógica del programa.
- ***Pruebas de integración:*** comprueban la combinación de las distintas partes de la aplicación. De esta forma se comprueba si el sistema funciona correctamente en conjunto.
- ***Pruebas de caja negra:*** son pruebas que se centran en los requerimientos establecidos en la fase de análisis y en la funcionalidad de la aplicación.
- ***Pruebas del sistema:*** comprueban el sistema completo como un todo y abarcan todas las partes combinadas del sistema.

6.2.1 Pruebas de caja blanca

Las pruebas de caja blanca tratan de encontrar fallos en el código de la aplicación, buscando posibles comportamientos inadecuados de las funciones que componen cada módulo.

Este subapartado se centrará en la realización de pruebas sobre las funciones más relevantes del modelo ya que en éste se ha implementado toda la lógica del sistema. Buscaremos errores, comprobaremos el funcionamiento de las funciones con valores normales, valores límites o valores erróneas.

Tengo que destacar que a medida que he ido desarrollando el código se ha tratado de recorrer todos los caminos de ejecución de cada una de las funciones implementadas. He testado cada una de las operaciones lógicas, he comprobado la definición y uso de las variables y sobre todo los bucles presentes en el código.

En el Anexo L podemos ver la Tabla 26 que contiene las funciones del modelo su descripción y la valoración respecto a las pruebas.

Como se puede ver en los resultados de la Tabla 26 (ver Anexo L) las pruebas fueron satisfactorias ya que se localizaron errores en algunas funciones. Todos los errores encontrados han sido corregidos.

Tras haber corregido los errores se relanzan nuevamente las baterías de pruebas y en esta ocasión ya no se encuentran más errores.

6.2.2 Pruebas de integración y de sistema

Con la realización de estas pruebas se trata de encontrar errores en las interfaces de los módulos, para poder garantizar que funcionan correctamente y que la comunicación entre las clase sea correcta.

6.2.2.1 Pruebas de caja negra

La finalidad de estas pruebas es comprobar que los requisitos funcionales se han cumplido. Estas pruebas permiten contener conjunto de condiciones de entrada que ejerciten completamente los requisitos funcionales definidos en la fase de análisis.

Para la realización de estas pruebas se ha comprobado uno a uno si todos los requisitos funcionales habían sido cubiertos.

En el Anexo J se representa los casos de pruebas de cada requisito en tablas.

Como es de esperar, las pruebas realizadas anteriormente han sido ejecutadas en el sistema final, es por ello que el resultado de casi todas ellas sea el esperado. Pero esto no fue así durante el desarrollo del trabajo. Durante este desarrollo se han ido encontrado resultados incorrectos, causados principalmente a errores de programación, de diseño, o de planteamiento que se han ido corrigiendo durante la implementación del mismo.

6.2.2.2 Pruebas de sistema

Para la realización de esta prueba se usa la placa que se ha indicado al inicio de este apartado, y a la cual se ha añadido un circuito para poder usar el sensor de luz tal y como se puede ver en la Ilustración 33.

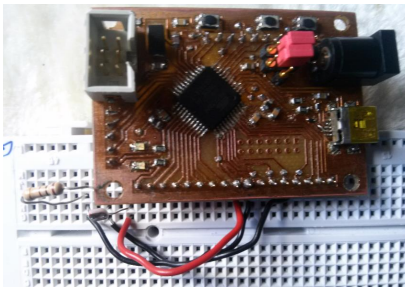

Ilustración 33: Circuito Sensor de luz

En el Anexo K se indica las pruebas que se han realizado para comprobar el funcionamiento correcto del sistema completo.

Tras las pruebas realizadas, los resultados obtenidos fueron satisfactorios y se ha conseguido obtener lo que realmente se deseaba. Con estas últimas pruebas se ha comprobado que el sistema desarrollado ha conseguido los objetivos que se plantearon al inicio de este trabajo y que finalmente los cumple.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

En este trabajo se ha conseguido implementar una interfaz gráfica capaz de programar las funcionalidades, que el usuario haya definido previamente en la aplicación, a una plataforma hardware a través del protocolo de comunicación USB por lo que se ha logrado el objetivo del presente proyecto.

Con este sistema conseguimos facilitar otra herramienta más a la comunidad educativa que les permita reforzar las enseñanzas de la robótica a partir de elementos más intuitivos y sencillos de usar que además despiertan el interés de los estudiantes de colegio hasta la Universidad. Esta herramienta será de gran ayuda para el aprendizaje de la robótica tanto si el usuario que la utiliza parte desde el desconocimiento de la materia como si se maneja en los niveles más avanzados. Esto será posible ya que la aplicación tendrá disponible dos niveles de dificultad, básico y avanzado. En el nivel básico, el usuario no tendrá que entrar en la implementación de nuevas funcionalidades ni tener conocimiento de las librerías que dejamos a disposición de aquellos usuarios con conocimiento de programación de microcontroladores en C. En el nivel avanzado, el usuario podrá definir nuevas funciones en esta librería y tendrá la posibilidad de darlas de alta o de baja desde la herramienta.

Además, este programa tal y como se ha comentado a lo largo de este documento, permite ver los puertos de comunicación disponibles por el ordenador con la posibilidad de seleccionar el puerto de comunicación con el robot.

En lo que se refiere al código de la aplicación, hemos intentado al máximo implementarlo de tal forma que sea lo más legible posible intentando comentar todos aquellos aspectos que sean importantes para comprender el funcionamiento tanto del programa, como de los elementos que lo componen. Además, se ha procedido a realizar varias revisiones con el fin de minimizar los recursos que utiliza el programa.

Por último, a nivel personal, la realización de este trabajo me ha permitido aprender de manera más autodidáctica la implementación de programas para aplicaciones de escritorio. Me gustaría añadir que durante el desarrollo de mi TFG he repasado gran cantidad de conceptos que he ido aprendiendo a lo largo de la carrera pero también he tenido que tomar decisiones de análisis, diseño, implementación, etc. Es la primera vez que he realizado un proyecto el cual no viene definido por un enunciado en concreto.

7.2 Trabajo futuro

Tras haber visto los objetivos alcanzados, se va a comentar alguna de las mejoras y ampliaciones que se podrán realizar como trabajo futuro con respecto a la herramienta implementada. Se han identificado las siguientes posibles líneas de mejora de la aplicación:

- Se podría mejorar la herramienta ampliándola para que pueda hacer uso de más periférico y no solo los que se han definido en el ***PFC [6].***

- Por otro lado, la herramienta simplemente nos permite testear de forma independiente cada uno de los periféricos presentes en el robot, con lo cual, una posible mejora sería incorporar a la herramienta la posibilidad de que nos muestre de forma gráfica la posición actual y el movimiento del robot, la incidencia de luz, la temperatura, etc. Esto nos permitirá saber el estado en el que se encuentra el robot y el funcionamiento de cada uno de los periféricos sin tener que testear de forma independiente cada uno de ellos.

- El sistema no muestra la información de los pines del microcontrolador correspondientes a cada uno de los periféricos conectados al micro. Una posible mejora sería que en el nivel de dificultad avanzado, se mostrase los pines del micro a los que está conectado cada uno de los periféricos y los puertos correspondientes.

- La herramienta almacena la información en ficheros de texto, sería recomendable que dicha información se almacenase en una base de datos ya que sería mucho más flexible a la hora de recuperar y almacenar la información.

- Por otro lado, sería posible la creación de una opción que permitiese ejecutar un programa de comandos en su totalidad, y que dicho programa pudiera editarse, cambiarse, grabarse y volverse a ejecutar de principio a fin. En caso de fallo o de incompatibilidades en la secuencia de órdenes, una pantalla de error nos informaría de la posible causa y el número de línea en el que dicho fallo se encuentra para poder corregirlo rápidamente.

Referencias

[1] Ardublock: Aplicación para introducir a niños en la programación de la robótica: http://es.slideshare.net/cantabrobots30/ardublock-39508070

[2] Blockly (Creado por Goocle):
http://www.genbetadev.com/herramientas/google-blockly-un-lenguaje-visual-para-aprender-a-programar
http://www.desarrolloweb.com/de_interes/blocky-aprender-programar-google-code-7127.html

[3] Scratch (desarrollada por el grupo Lifelong Kindergarten del MIT Media Lab):
http://www.xataka.com/aplicaciones/como-iniciar-a-un-nino-en-la-programacion-desde-cero-con-scratch
http://vps34736.ovh.net/S4A/s4a-manual.pdf
http://solorobotica.blogspot.com.es/2012/04/s4a-scratch-para-arduino.html

[4] Enseñar a programar a los niños en el colegio debería se una prioridad.
http://www.adslzone.net/2015/09/24/ensenar-a-programar-a-los-ninos-en-el-colegio-deberia-ser-una-prioridad/
http://www.xataka.com/especiales/ninos-y-programacion-consejos-y-recursos-para-que-este-verano-se-inicien
http://www.eldiario.es/turing/Ninos-programadores-ensenanza-programacion-escuelas_0_293970921.html

[5] Asignatura de Programación y robótica en la ESO:
http://www.abc.es/espana/madrid/abci-asignatura-programacion-y-robotica-extendera-toda-proximo-curso-201602272127_noticia.html
http://ccaa.elpais.com/ccaa/2014/09/03/madrid/1409772225_352560.html

[6] Proyecto fin de carrera de Jaime Díaz García: "Plataforma modular de aprendizaje para robótica móvil" – PFC en proceso aún no finalizado.

[7] Página oficial del robot Arduino: http://arduino.cc/en/main/robot

[8] Plataformas para el aprendizaje de la programación:
http://codigo21.educacion.navarra.es/familias/recursos/

[9] Robot Dot and Dash:
https://www.amazon.es/Wonder-Workshop-Pack-robots-educativos/dp/B00X2PJQZ0

[10] Robots Educativos: Apredizaje con Dot and Dash
http://www.vicensvives.com/robots-educativos/

[11] App Blockly:
http://www.genbetadev.com/herramientas/google-blockly-un-lenguaje-visual-para-aprender-a-programar

[12] Aplicación mblock para aprender a programar robots:

https://makeblock.es/tutoriales/aprender_programar_robots_con_mblock/
https://www.makeblock.es/productos/robot_educativo_mbot/

[13] Scratch, un proyecto gratuito del MIT que enseña a los niños a programar:
http://hipertextual.com/2014/05/scratch

[14] Arduino (Plataforma de desarrollo de circuitos electrónico):
http://www.xataka.com/especiales/guia-del-arduinomaniaco-todo-lo-que-necesitas-saber-sobre-arduino

[15] Ardublock, Herramienta de programación visual por bloques:
http://blog.ardublock.com/
http://ardublock.blogspot.com.es/2014/07/instalacion-ardublock.html
http://es.slideshare.net/cantabrobots30/ardublock-3950807

[16] S4A plataforma para programar una placa de Arduino:
http://s4a.cat/index_es.html
https://tecnopujol.wordpress.com/s4a/
http://recursostic.educacion.es/observatorio/web/fr/software/software-educativo/1018-monograficodesarrollos-de-scratch-para-robotica-enchanting-y-s4a?start=3

[17] Definición USB:
https://es.wikipedia.org/wiki/Universal_Serial_Bus

[18] Definición del dispositivo Hub:
https://es.wikipedia.org/wiki/Hub_USB

[19] CDC: Communication Device Class:
http://www.keil.com/pack/doc/mw/USB/html/_c_d_c.html

[20] Herramienta NetBeans:
https://netbeans.org/

[21] James Gosling , "The Java language Specification". Sun Microsystems, 2000.
https://books.google.es/books?hl=es&lr=&id=Ww1B9O_yVGsC&oi=fnd&pg=PA1&dq=#v=onepage&q&f=false

[22] USB DFU Booltloader Datasheet:
http://www.atmel.com/Images/doc7618.pdf

[23] Download the USB DFU bootloader hex files:
http://www.atmel.com/devices/atmega32u4.aspx?tab=documents//www.atmel.com/Images/doc7618.pdf
[24] Herramienta Atmel Studio:
http://www.atmel.com/tools/ATMELSTUDIO.aspx?tab=overview
[25] Librería JSSC para la comunicación por el puerto serie:
https://github.com/scream3r/java-simple-serial-connector
[26] Proyecto fin de carrera de Alfredo Manuel Castro:
http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20140930AlfredoManuelCastroCuartero.pdf

Glosario

API	Application Programming Interface
USB	Universal Serial Bus
CDC	Communication Device Class

Anexos

A Descripción de los Casos de Uso

A.1 Usuario

[UC-0001]: Seleccionar puerto de comunicación

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario debe seleccionar el puerto de comunicación para intercambiar información entre el ordenador y el robot.

Precondiciones: está seleccionada la opción "Cargar periféricos".

Garantía de éxito (Poscondición): se muestra el puerto de comunicación utilizado para establecer la comunicación entre el ordenador y el robot.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario solicita "Cargar periféricos".
2- La aplicación muestra una ventana con el puerto de comunicación detectado.
3- El caso de uso finaliza cuando el usuario selecciona el puerto de comunicación.

Extensiones (Flujos alternativos):

2.a El sistema no detecta ningún puerto de comunicación.
2.a.1 En este caso el sistema informa al usuario que no se ha detectado ningún puerto de comunicación.

Frecuencia de ocurrencia: baja. El usuario solo seleccionará el puerto de comunicación solo cuando se cargar por primera vez los periféricos.

Temas abiertos: no hay temas abiertos.

[UC-0002]: Cargar periférico

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere cargar los periféricos del robot en la aplicación.

Precondiciones: el usuario debe haber seleccionado el puerto de comunicación para establecer conexión entre el pc y el robot, y seleccionado la opción “Cargar periféricos”.

Garantía de éxito (Poscondición): se cargan los periféricos en la aplicación y el usuario puede hacer uso de los mismos.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario solicita “Cargar periféricos”.
2- La aplicación muestra el puerto de comunicación para establecer la conexión con el robot y solicitar los periféricos disponibles.
3- El caso de uso finaliza cuando la aplicación carga los periféricos en la aplicación.

Extensiones (Flujos alternativos):

2.a E sistema no detecta ningún puerto de comunicación.

2.a.1 En este caso el sistema informa al usuario que no se ha detectado ningún puerto de comunicación.

Frecuencia de ocurrencia: baja. Por ahora el usuario solo podrá cargar los periféricos al principio, por lo que se supone que no realizará esta acción con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0003]: Ver detalle periférico

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere ver en detalle la información de cada uno de los periféricos disponibles en el robot.

Precondiciones: el usuario debe haber cargado previamente los periféricos en la aplicación y seleccionado la opción “Ver detalles”.

Garantía de éxito (Poscondición): se muestra la información detallada de cada uno de los periféricos.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario solicita “Ver detalles”.
2- El caso de uso finaliza cuando la aplicación muestra la información detallada del periférico seleccionado.

Extensiones (Flujos alternativos): no tiene flujos alternativos.

Frecuencia de ocurrencia: alta. El usuario realizará con frecuencia este caso de uso ya que necesitará consultar la descripción de cada uno de los periféricos.

Temas abiertos: no hay temas abiertos.

[UC-0004]: Testeo de los periféricos

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario desea comprobar el correcto funcionamiento de los periféricos disponibles en la aplicación.

Precondiciones: se debe haber seleccionado el puerto de comunicación y la opción "Testear periféricos".

Garantía de éxito (Poscondición): el usuario comprueba el funcionamiento de cada uno de los periféricos.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario selecciona la opción "Testear periféricos".
2- La aplicación muestra una ventana con todos los periféricos disponibles para comprobar su funcionamiento.
3- El caso de uso finaliza cuando el usuario realiza el testeo del periférico.

Extensiones (Flujos alternativos):

2.a El sistema no ha detectado ningún puerto de comunicación.
2.a.1 En este caso el sistema informa al usuario que no hay puerto de comunicación y no se abre la ventana para testear los periféricos.

Frecuencia de ocurrencia: Alta. Un sistema de este tipo tiene como objetivo añadir los periféricos al diseño elaborado en el entorno gráfico, por lo que testear cada periférico para comprobar que funciona correctamente se convertirá en una funcionalidad muy solicitada en la aplicación.

Temas abiertos: no hay temas abiertos.

[UC-0005]: Añadir periférico

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario desea añadir un periférico al diseño elaborado en el entorno gráfico.

Precondiciones: se debe haber seleccionado la opción "Añadir periférico"

Garantía de éxito (Poscondición): el sistema crea el nuevo periférico en el entorno gráfico.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario selecciona "Añadir periférico".
2- El sistema crea un nuevo periférico y lo añade al diseño elaborado en el entorno gráfico.
3- El caso de uso finaliza.

Extensiones (Flujos alternativos):

2.a El usuario añade un periférico al diseño elaborado en el entorno gráfico.

2.a.1 En este caso se actualiza los periféricos presentes en el entorno gráfico.

Frecuencia de ocurrencia: Muy alta. Una de las funciones de este sistema es añadir los periféricos a un diseño elaborado en el entorno gráfico, por lo que el añadir periféricos se convertirá en una funcionalidad que se ejecutará con mucha frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0006]: Eliminar periférico

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario desea eliminar un periférico del diseño elaborado en el entorno gráfico.

Precondiciones: se debe haber seleccionado la opción "Eliminar"

Garantía de éxito (Poscondición): el sistema elimina el periférico seleccionado del entorno gráfico.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario selecciona "Eliminar".
2- El sistema elimina, del entorno gráfico, el periférico seleccionado.
3- El caso de uso finaliza con la eliminación del periférico del entorno gráfico.

Extensiones (Flujos alternativos):

2.a El usuario elimina el periférico del diseño elaborado en el entorno gráfico.

2.a.1 En este caso se actualiza los periféricos presentes en el entorno gráfico.

Frecuencia de ocurrencia: Muy alta. El usuario utilizará esta funcionalidad cada vez que desee eliminar un periférico del diseño elaborado en el entorno gráfico.

Temas abiertos: no hay temas abiertos.

[UC-0007]: Eliminar todos los periféricos

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario desea eliminar todos los periféricos del diseño elaborado en el entorno gráfico.

Precondiciones: se debe haber seleccionado la opción “Eliminar todo”.

Garantía de éxito (Poscondición): el sistema elimina todos los periféricos del entorno gráfico.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario selecciona “Eliminar todo”.
2- El sistema elimina, del entorno gráfico, todos los periféricos presentes en el entorno gráfico.
3- El caso de uso finaliza con la eliminación de los periféricos del entorno gráfico.

Extensiones (Flujos alternativos):

2.a El usuario elimina todos los periféricos del entorno gráfico.

2.a.1 En este caso se actualiza los periféricos presentes en el entorno gráfico.

Frecuencia de ocurrencia: Alta. El usuario utilizará esta funcionalidad cada vez que desee eliminar todos los periféricos presente en el diseño desarrollado en el entorno gráfico.

Temas abiertos: no hay temas abiertos.

[UC-0008]: Seleccionar situación

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere seleccionar la situación en la que se encuentra el robot.

Precondiciones: se debe haber seleccionado la opción "Seleccionar acciones del robot"

Garantía de éxito (Poscondición): el sistema almacena las situaciones seleccionadas por el usuario.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario selecciona la opción "Seleccionar acciones del robot".
2- El sistema cargar de forma dinámica las posibles situaciones en la que se encuentra el robot según los periféricos presentes en el diseño elaborado en el entorno gráfico.
3- El caso de uso finaliza cuando el usuario ha seleccionado la situación deseada.

Extensiones (Flujos alternativos):

2.a. El sistema cargar de forma dinámica las posibles situaciones en la que se podrá encontrar el robot.

2.a.1 En este caso el sistema almacenará las situaciones seleccionadas por el usuario.

Frecuencia de ocurrencia: Muy alta. Un sistema de este tipo tiene como objetivo principal seleccionar las situaciones en la que se encontrará el robot, por lo que este caso de uso se convertirá en una funcionalidad muy solicitada en la aplicación.

Temas abiertos: no hay temas abiertos.

[UC-0009]: Seleccionar acción

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere seleccionar las acciones que quiere que realice el robot en una situación.

Precondiciones: se debe haber seleccionado una situación en la que se encuentra el robot.

Garantía de éxito (Poscondición): el sistema almacena las acciones seleccionadas por el usuario.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario selecciona una de las posibles situaciones disponibles en el sistema.
2- El sistema cargar de forma dinámica las posibles acciones que podrá realizar el robot según los periféricos presentes en el diseño elaborado en el entorno gráfico.
3- El caso de uso finaliza cuando el usuario ha seleccionado las acciones deseadas.

Extensiones (Flujos alternativos):

2.a. El sistema cargar de forma dinámica las posibles acciones que podrá realizar el robot.

2.a.1 En este caso el sistema almacenará las acciones seleccionadas por el usuario.

Frecuencia de ocurrencia: Muy alta. Un sistema de este tipo tiene como objetivo principal seleccionar las acciones que realizará el robot, por lo que este caso de uso se convertirá en una funcionalidad muy solicitada en la aplicación.

Temas abiertos: no hay temas abiertos.

[UC-0010]: Cargar funcionalidad al robot

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario quiere cargar al robot la funcionalidad obtenida como las combinaciones de la situación y acción seleccionada.

Precondiciones: se debe haber seleccionado la opción "Seleccionar las acciones del robot".

Garantía de éxito (Poscondición): el sistema carga al robot las funcionalidades definidas por el usuario.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario ha seleccionado la opción "Cargar programa al robot".
2- El sistema carga al robot las funciones previamente definidas.
3- El caso de uso finaliza cuando se ha cargado correctamente las funciones definidas en el robot.

Extensiones (Flujos alternativos):

2.a El sistema carga las funciones previamente definidas:

2.a.1 En este caso si no hay puerto de comunicación con el robot el sistema informará al usuario de ello para que posteriormente revise la conexión entre el ordenador y robot.

Frecuencia de ocurrencia: Alta. Un sistema de este tipo tiene como objetivo principal elaborar un diseño en un entorno gráfico, seleccionar las acciones deseadas por el usuario y cargarlas en el robot, por lo que cargar las funciones previamente seleccionadas se convertirá en una funcionalidad muy solicitada en la aplicación.

Temas abiertos: no hay temas abiertos.

[UC-0011]: Guardar diseño

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario desea guardar el diseño elaborado en el entorno gráfico.

Precondiciones: se debe haber seleccionado la opción "Guardar" o "Guardar como"

Garantía de éxito (Poscondición): el sistema guarda un fichero con todo el diseño en el directorio seleccionado por el usuario.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario ha seleccionado la opción "Guardar" o "Guardar Como".
2- El sistema guarda el diseño en el directorio seleccionado por el usuario.
3- El caso de uso finaliza cuando el usuario ha guardado correctamente el diseño.

Extensiones (Flujos alternativos):

2.a El sistema guarda un fichero con el diseño en el directorio seleccionado por el usuario:

2.a.1 En este caso si ya existe un fichero con el mismo nombre, se indicará al usuario si desea reemplazarlo o no.

Frecuencia de ocurrencia: Alta. El usuario podrá guardar su diseño en el momento que desee, por lo que está función podrá ser ejecutada con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0012]: Abrir diseño

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario desea abrir un diseño guardado previamente.

Precondiciones: se debe haber seleccionado la opción "Abrir".

Garantía de éxito (Poscondición): el sistema carga un fichero existente con todo el diseño.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario ha seleccionado la opción "Abrir".
2- El sistema carga el diseño en el entorno de trabajo.
3- El caso de uso finaliza cuando se ha cargado el diseño en el entorno de trabajo.

Extensiones (Flujos alternativos):

2.a El sistema carga el diseño en el entorno de trabajo:
2.a.1 En este caso si ya hay un diseño en el entorno gráfico en el momento de abrir uno nuevo el sistema indicará al usuario que guarde el diseño actual.

Frecuencia de ocurrencia: baja. El usuario podrá abrir su diseño en un momento puntual, por lo que está función no será ejecutada con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0013]: Dar de alta un periférico

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere añadir un nuevo periférico para poderlo usar en la aplicación.

Precondiciones: haber seleccionado la opción "Dar de alta un periférico".

Garantía de éxito (Poscondición): el sistema almacena en un fichero el periférico dado de alta desde la aplicación.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario ha seleccionado la opción "Dar de alta periférico".
2- El usuario selecciona el periférico que quiere dar de alta.

3- El caso de uso finaliza cuando el usuario ha dado de alta el periférico y se ha almacenado en un fichero de texto.

Extensiones (Flujos alternativos):

2.a El usuario selecciona el periférico que quiere dar de alta:
2.a.1 En este caso si el periférico que desea dar de alta el usuario ya existe en la aplicación, el sistema indicará de ello al usuario.

Frecuencia de ocurrencia: baja. No es una acción que el usuario lo realizará con frecuencia ya que el dar de alta un nuevo periférico es una acción que se realizará de forma puntual y no con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0014]: Dar de baja un periférico

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere eliminar un periférico de la aplicación.

Precondiciones: haber seleccionado la opción "Dar de baja un periférico".

Garantía de éxito (Poscondición): el sistema elimina de un fichero el periférico que ha dado de baja.

Escenario principal de éxito:

1- El caso de uso comienza cuando el usuario ha seleccionado la opción "Dar de baja periférico".
2- El usuario selecciona el periférico que quiere dar de baja.
3- El caso de uso finaliza cuando el usuario ha dado de baja el periférico y se ha eliminado del fichero de texto que lo almacenaba.

Extensiones (Flujos alternativos):

2.a El usuario selecciona el periférico que quiere dar de baja:
2.a.1 En este caso el periférico se elimina del fichero correspondiente.

Frecuencia de ocurrencia: baja. No es una acción que el usuario lo realizará con frecuencia ya que el dar de baja un periférico es una acción que se realizará de forma puntual y no con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0015]: Dar de alta una situación del robot

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere dar de alta una nueva situación en la que se podrá encontrar el robot.

Precondiciones: La aplicación tiene que estar en el nivel de dificultad avanzado y haber seleccionado la opción "Dar de alta una situación".

Garantía de éxito (Poscondición): el sistema almacena en un fichero la situación dada de alta desde la aplicación.

Escenario principal de éxito:

1- La aplicación tiene que estar en el modo Avanzado
2- El caso de uso comienza cuando el usuario ha seleccionado la opción "Dar de alta una situación".
3- El usuario seleccionara el periférico e introducirá la situación y función deseada.
4- El caso de uso finaliza cuando el usuario ha dado de alta la situación y se ha almacenado en un fichero de texto.

Extensiones (Flujos alternativos):

3.a El usuario seleccionará el periférico e introducirá la situación y función deseada.

3.a.1 En este caso si la situación que desea dar de alta ya existe en la aplicación, el sistema indicará de ello al usuario.

Frecuencia de ocurrencia: baja. No es una acción que el usuario lo realizará con frecuencia ya que el dar de alta una nueva situación es una acción que se realizará de forma puntual y no con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0016]: Dar de baja una situación del robot

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere dar de baja una situación en la que se podrá encontrar el robot.

Precondiciones: La aplicación tiene que estar en el nivel de dificultad avanzado y haber seleccionado la opción "Dar de baja una situación".

Garantía de éxito (Poscondición): el sistema elimina de un fichero la situación que ha dado de baja el usuario.

Escenario principal de éxito:

1- La aplicación tiene que estar en el modo Avanzado
2- El caso de uso comienza cuando el usuario ha seleccionado la opción "Dar de baja una situación".
3- El usuario selecciona la situación que quiere dar de baja.
4- El caso de uso finaliza cuando el usuario ha dado de baja la situación y se elimina del fichero de texto correspondiente.

Extensiones (Flujos alternativos):

3.a El usuario selecciona la situación que quiere dar de baja:
3.a.1 En este caso la situación se elimina del fichero correspondiente.

Frecuencia de ocurrencia: baja. No es una acción que el usuario lo realizará con frecuencia ya que el dar de baja una situación es una acción que se realizará de forma puntual y no con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0017]: Dar de alta una acción

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere eliminar de la aplicación la situación correspondiente a un periférico del robot.

Precondiciones: La aplicación tiene que estar en el nivel de dificultad avanzado y haber seleccionado la opción "Dar de alta una acción".

Garantía de éxito (Poscondición): el sistema almacena en un fichero la acción dada de alta desde la aplicación.

Escenario principal de éxito:

1- La aplicación tiene que estar en el modo Avanzado
2- El caso de uso comienza cuando el usuario ha seleccionado la opción "Dar de alta una acción".
3- El usuario seleccionara el periférico e introducirá la acción y función deseada.
4- El caso de uso finaliza cuando el usuario ha dado de alta la situación y se ha almacenado en un fichero de texto.

Extensiones (Flujos alternativos):

3.a El usuario seleccionará el periférico e introducirá la acción y función deseada.

3.a.1 En este caso si la acción que desea dar de alta ya existe en la aplicación, el sistema indicará de ello al usuario.

Frecuencia de ocurrencia: baja. No es una acción que el usuario lo realizará con frecuencia ya que el dar de alta una nueva acción es algo que se realizará de forma puntual y no con frecuencia.

Temas abiertos: no hay temas abiertos.

[UC-0018]: Dar de baja una acción

Actor primario: usuario de la aplicación.

Interesados y objetivos: el usuario que quiere eliminar de la aplicación la acción correspondiente a un periférico del robot.

Precondiciones: La aplicación tiene que estar en el nivel de dificultad avanzado y haber seleccionado la opción "Dar de baja una acción".

Garantía de éxito (Poscondición): el sistema elimina de un fichero la acción que ha dado de baja el usuario.

Escenario principal de éxito:

1- La aplicación tiene que estar en el modo Avanzado
2- El caso de uso comienza cuando el usuario ha seleccionado la opción "Dar de baja una acción".
3- El usuario selecciona la acción que quiere dar de baja.
4- El caso de uso finaliza cuando el usuario ha dado de baja la acción y se elimina del fichero de texto correspondiente.

Extensiones (Flujos alternativos):

3.a El usuario selecciona la acción que quiere dar de baja:
3.a.1 En este caso la acción se elimina del fichero correspondiente.

Frecuencia de ocurrencia: baja. No es una acción que el usuario lo realizará con frecuencia ya que el dar de baja una acción es algo que se realizará de forma puntual y no con frecuencia.

Temas abiertos: no hay temas abiertos.

B Requisitos funcionales del Subsistema Usuario

[RF1-U1]
Detectar puerto de comunicación
Para poder establecer comunicación entre el ordenador y el robot, es necesario que el sistema sea capaz de detectar el puerto de comunicación por el cual se enviarán y se recibirán las ordenes al y desde el robot.

[RF2-U2]
Leer y cargar los periféricos del robot
El sistema deberá proveer al usuario la funcionalidad para poder leer y cargar los periféricos de los que dispone el robot.

[RF3-U3]
Ver el número máximo de cada periférico
Cuando el usuario haya cargado los periféricos en la aplicación, el sistema deberá mostrar el número máximo de cada uno de los periféricos.

[RF4-U4]
Ver detalle
El sistema deberá proveer al usuario la funcionalidad para poder ver la descripción de cada uno de los periféricos disponibles en la aplicación.

[RF5-U5]
Testeo de los periféricos
El sistema permitirá al usuario realizar un testeo de cada uno de los periféricos disponible y cargados previamente en la aplicación.

[RF6-U6]
Añadir periférico
El sistema permitirá al usuario añadir un periférico al diseño elaborado en el entorno gráfico.

[RF7-U7]
Eliminar periférico
El sistema permitirá al usuario eliminar uno o todos los periféricos del diseño elaborado en el entorno gráfico.

[RF8-U8]
Cargar situaciones
El sistema permitirá cargar de forma dinámica las posibles situaciones en las que se podrá encontrar el robot dependiendo de los periféricos que se hayan añadido previamente al diseño gráfico.

[RF9-U9]
Seleccionar situación
El sistema permitirá al usuario seleccionar las posibles situaciones dependiendo de la funcionalidad que se desea que realice el robot.

[RF10-U10]
Cargar acciones
El sistema permitirá cargar de forma dinámica las acciones que podrá realizar el robot en cada una de las situaciones seleccionadas.

[RF11-U11]
Seleccionar acción
El sistema permitirá al usuario seleccionar las posibles acciones asociándolas a una de las situaciones disponibles en la aplicación.

[RF12-U12]
Mostrar funcionalidad
El sistema deberá mostrar un resumen con todas las funcionalidades que llevará a cabo el robot.

[RF13-U13]
Crear programa
El sistema deberá crear el programa en lenguaje C para posteriormente ser compilado y obtener el programa hexadecimal que se cargará al robot. Esto es transparente al usuario, es decir, el usuario no tiene idea de esta operación.

[RF14-U14]
Cargar a robot
El sistema deberá traducir las funcionalidades seleccionadas a un programa en C que posteriormente será compilado para obtener el programa hexadecimal que se cargará al robot. Del mismo modo, esto es transparente al usuario, es decir, el usuario no tiene idea de esta operación.

[RF15-U15]
Guardar
El sistema deberá proporcionar al usuario la posibilidad de guardar el diseño elaborado en el entorno gráfico.

[RF16-U16]
Abrir
El sistema deberá proporcionar al usuario la posibilidad de poder cargar en el entorno gráfico un diseño guardado previamente.

[RF17-U17]
Dar de alta un periférico
El sistema deberá proporcionar al usuario la posibilidad de poder dar de alta un periférico desde la aplicación.

[RF18-U18]
Dar de baja un periférico
El sistema deberá proporcionar al usuario la posibilidad de poder dar de baja un periférico desde la aplicación.

[RF19-U19]
Dar de alta una situación

El sistema deberá proporcionar al usuario la posibilidad de poder dar de alta, desde la aplicación, las posibles situaciones en las que se podrá encontrar el robot.

[RF20-U20]
Dar de baja una situación
El sistema deberá proporcionar al usuario la posibilidad de poder dar de baja, desde la aplicación, las posibles situaciones en las que se podrá encontrar el robot.

[RF21-U21]
Dar de alta una acción
El sistema deberá proporcionar al usuario la posibilidad de poder dar de alta, desde la aplicación, las posibles acciones que podrá llevar acabo el robot.

[RF22-U22]
Dar de baja una acción
El sistema deberá proporcionar al usuario la posibilidad de poder dar de baja, desde la aplicación, las posibles acciones que podrá llevar a cabo el robot.

C Descripción de las funciones de la API

En este anexo se va a describir todas las funciones de la API que se han mostrado en la fase de diseño de software.

- checkSerialPort
    - **Descripción:** permite comprobar si hay conectado algún puerto de comunicación.
    - **Entrada:**
        - (Ninguna)
    - **Salida:**
        - 0 → todo ha ido bien entonces muestra la ventana para seleccionar le puerto de comunicación.
        - 1 → no se puede abrir el dispositivo USB.
        - 2 → no se puede escribir en el dispoisitivo USB.
    - **Requisitos:** REF1-U1.

- readPeripheralsInRobot
    - **Descripción:** permite leer los periféricos que dispone el robot.
    - **Entrada:**
        - (Ninguna)
    - **Salida:**
        - true → se lee correctamente los periféricos disponibles.
        - false → problemas en la lectura de los periféricos disponibles.
    - **Requisitos:** RF1-U1, RF2-U2.

- setNumMaxPeripherals
    - **Descripción:** permite almacenar el número máximo de cada periférico.
    - **Entrada:**
        - (Ninguna)
    - **Salida:**
        - true → se asigna el número máximo a cada periférico.
        - false → no se asigna el número máximo a cada periférico.
    - **Requisitos:** RF2-U2, RF3-U3.

- viewDetailsPeripheral

    - **Descripción:** permite obtener una pequeña descripción de cada periférico.

    - **Entrada:**
        - (Ninguna)

    - **Salida:**
        - true → se muestra la descripción del periférico.
        - false → no es posible mostrar la descripción del periférico.

    - **Requisitos:** RF2-U2, RF4-U4.

- openTestPeripheralsFrame

    - **Descripción:** abre una ventana que permite testear cada uno de los periféricos disponibles.

    - **Entrada:**
        - (Ninguna)

    - **Salida:**
        - true → se inicia la ventana para testear cada uno de los periféricos.
        - false → no se inicia la ventana de testeo de periféricos.

    - **Requisitos:** RF1-U1, RF5 -U5.

- addPeripheralWorkspace

    - **Descripción:** permite añadir un periférico en el entorno gráfico.

    - **Entrada:**
        - (Ninguna)

    - **Salida:**
        - true → se añade el periférico al entorno gráfico.
        - false → no se añade el periférico al entorno de trabajo.

    - **Requisitos:** RF1-U1, RF6 –U6.

- deletePeripheralWorkspace

    - **Descripción:** permite eliminar un periférico del entorno gráfico.

- **Entrada:**
  - peripheral: periférico que se desea eliminar del entorno gráfico.

- **Salida:**
  - true → se elimina el periférico del entono gráfico.
  - false → no se elimina el periférico del entorno gráfico.

- **Requisitos:** RF7–U7.

- deleteAllPeripheralsWorkspace

  - **Descripción:** permite eliminar todos los periféricos del entorno gráfico.

  - **Entrada:**
    - (Ninguno).

  - **Salida:**
    - true → se eliminan todos los periféricos del entono gráfico.
    - false → los periféricos no se elimina del entorno gráfico.

  - **Requisitos:** RF7–U7.

- readSituationOfRobot

  - **Descripción:** permite leer las situaciones en las que puede estar el robot.

  - **Entrada:**
    - (Ninguno).

  - **Salida:**
    - true → se cargan correctamente las situaciones en el entorno gráfico.
    - false → las situaciones no se cargan en el entorno gráfico.

  - **Requisitos:** RF8–U8.

- addListenerSituations

  - **Descripción:** permite seleccionar una situación.

  - **Entrada:**
    - situación

  - **Salida:** (Ninguna)

  - **Requisitos:** RF9-U9.

- readActionsRobot
    - **Descripción:** permite leer las acciones que podrá realizar el robot.
    - **Entrada:**
        - (Ninguno).
    - **Salida:**
        - true → se cargan correctamente las acciones en el entorno gráfico.
        - false → las acciones no se cargan en el entorno gráfico.
    - **Requisitos:** RF10–U10.

- viewActionsRobot
    - **Descripción:** permite mostrar las acciones que podrá realizar el robot.
    - **Entrada:**
        - mapPeripheralAction.
    - **Salida:**
        - true → las acciones se muestran en el entorno de trabajo y el usuario las puede seleccionar.
        - false → las acciones no se muestran en el entorno gráfico.
    - **Requisitos:** RF11–U11.

- updatePanelActionsRobot
    - **Descripción:** permite mostrar un resumen de las funcionalidades seleccionadas por el usuario.
    - **Entrada:**
        - (Ninguno).
    - **Salida:**
        - true → se muestra un resumen con las funciones seleccionadas por el usuario.
        - false → no se muestra el resumen.
    - **Requisitos:** RF12–U12.

- createProgram

    - **Descripción:** permite crear el programa hexadecimal que posteriormente se cargará en el robot.

    - **Entrada:**
        - (Ninguno).

    - **Salida:**
        - true → se ha creado el programa correctamente.
        - false → el programa no se ha creado correctamente.

    - **Requisitos:** RF13–U13.

- loadHEXToMicro

    - **Descripción:** permite cargar el programa hexadecimal en el robot.

    - **Entrada:**
        - (Ninguno).

    - **Salida:**
        - true → se ha cargado el programa correctamente.
        - false → no se ha cargado el programa en el robot.

    - **Requisitos:** RF14–U14.

- doSaveAPEFile

    - **Descripción:** permite guardar el diseño elaborado en el entorno gráfico.

    - **Entrada:**
        - (Ninguno).

    - **Salida:**
        - true → se ha guardado el diseño correctamente.
        - false → el diseño no se ha guardado correctamente.

    - **Requisitos:** RF15–U15.

- doSaveAsAPEFile

    - **Descripción:** permite 'guardar como..' el diseño elaborado en el entorno gráfico.

    - **Entrada:**
        - (Ninguno).

    - **Salida:**
        - true → se ha guardado el diseño correctamente.
        - false → el diseño no se ha guardado correctamente.

    - **Requisitos:** RF15–U15.

- doOpenAPEFile

    - **Descripción:** permite abrir un diseño previamente guardado.

    - **Entrada:**
        - (Ninguno).

    - **Salida:**
        - true → se carga el diseño en el entorno gráfico.
        - false → no es posible cargar el diseño en el entorno gráfico.

    - **Requisitos:** RF16–U16.

- registerPeripheralInFile

    - **Descripción:** permite dar de alta un periférico en la aplicación.

    - **Entrada:**
        - perif: periférico.
        - numMaximo: número máximo de este tipo de periférico.

    - **Salida:**
        - true → se da de alta el periférico.
        - false → no se da de alta el periférico.

    - **Requisitos:** RF17–U17.

- deregisterPeripheralInFile

    - **Descripción:** permite dar de baja un periférico en la aplicación.

    - **Entrada:**
        - perif: periférico.
        - numMaximo: número máximo de este tipo de periférico.

    - **Salida:**
        - true → se da de baja el periférico.
        - false → no se da de baja el periférico.

    - **Requisitos:** RF18–U18.

- registerSituationInFile

    - **Descripción:** permite dar de alta una situación para un periférico.

    - **Entrada:**
        - peripheral: periférico.
        - situation: situación del periférico.
        - function: función correspondiente a la situación.

    - **Salida:**
        - true → se da de alta la situación.
        - false → no se da de alta la situación.

    - **Requisitos:** RF19–U19.

- deregisterSituationsInFile

    - **Descripción:** permite dar de baja una situación correspondiente a un periférico.

    - **Entrada:**
        - situation: situación que se va a eliminar.

    - **Salida:**
        - true → se da de baja la situación de un periférico.
        - false → no ha sido posible dar de baja la situación.

    - **Requisitos:** RF20–U20.

- registerActionInFile

  - **Descripción:** permite dar de alta una acción para un periférico.

  - **Entrada:**
    - peripheral: periférico.
    - action: acción del periférico.
    - function: función correspondiente a la acción.

  - **Salida:**
    - true → se da de alta la acción.
    - false → no se da de alta la acción.

  - **Requisitos:** RF21–U21.

- deregisterActionInFile

  - **Descripción:** permite dar de baja una acción correspondiente a un periférico.

  - **Entrada:**
    - action: acción que se va a eliminar.

  - **Salida:**
    - true → se da de baja la acción de un periférico.
    - false → no ha sido posible dar de baja la acción.

  - **Requisitos:** RF22–U22.

D Diseño conceptual de las ventanas

A continuación se muestra la estructura que tendrá cada una de las ventanas de nuestra interfaz gráfica.

- **Ventana de inicio :: *Ventana inicial***

  En la Ilustración 35 se observa la ventana principal de la aplicación. En ella se puede ver que la interfaz gráfica dispondrá de una barra de menú en la parte superior donde cada una de estos menús tendrán asociado un conjunto de opciones/submenús.

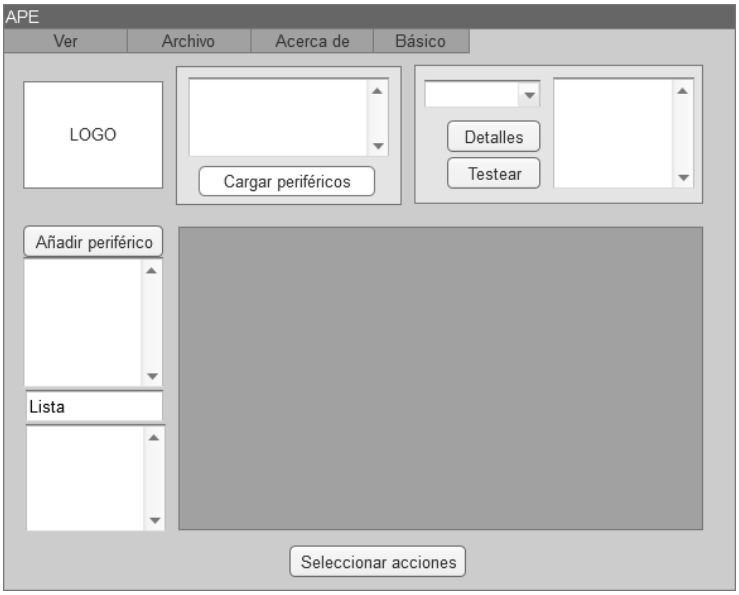

Ilustración 34: Diseño de la ventana principal

Además de la barra de menú, en la ventana principal se puede observar 5 botones los cuales se relacionan con la Carga de los periféricos en la interfaz gráfica, la posibilidad de testear los periféricos, ver la descripción de cada uno de los periféricos, añadir un periférico al entorno gráfico y seleccionar las funcionalidades que podrá tener el robot. También esta ventana tendrá áreas de texto en las cuales se indicará los periféricos con su número máximo en el robot, la descripción de cada periférico, los periféricos que se van a añadir al entorno gráfico y la lista de periféricos añadidos a este entorno. El entorno gráfico estará situado en el centro de la interfaz y acaparará gran parte de la ventana principal.

- **Barra de menú::*Menú básico/avanzado***

  Cuando se pulse sobre el menú 'Ver' se desplegarán dos opciones que indicarán el nivel de dificultad de la aplicación. Inicialmente la aplicación se iniciará con el nivel básico, dando la posibilidad al usuario solo de dar de alta o de baja periféricos en la aplicación. Si el usuario seleccionase la opción 'Nivel avanzado' en lugar de aparecer el menú 'Básico' aparecerá el menú 'Avanzado'

que tendrá asociado tres opciones: Periférico, Acción y Situación, y cada una de ellas tendrá asociado un submenú con las opciones ‘Dar de alta’ y ‘Dar de baja’.

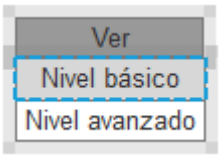

Ilustración 35: Menú Ver

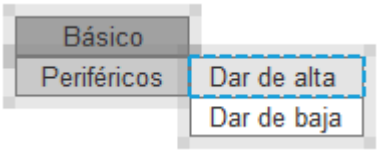

Ilustración 36: Menú nivel de dificultad Básico

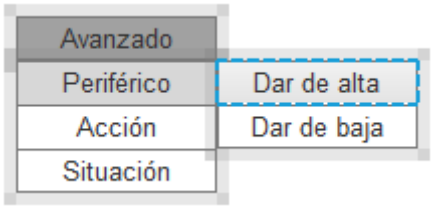

Ilustración 37: Menú nivel de dificultad Avanzado

Por otro lado, está el menú ‘Archivo’ que tendrá asociado las opciones necesarias para crear un nuevo diseño, guardar el diseño, abrir un diseño almacenado previamente y salir de la aplicación.

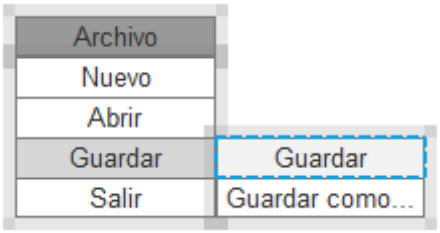

Ilustración 38: Menú Archivo

Finalmente está el menú ‘Acerca de’, que tendrá asociado una única opción la cual es mostrar la versión de la aplicación e información de la misma.

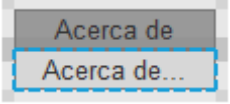

Ilustración 39: Menú Acerca de

- **Ventana para dar de alta un periférico::*Ventana alta periférico***

  Tanto en el nivel básico como el avanzado, el usuario podrá dar de alta o de baja los periféricos según la necesidad que tenga. Esta ventana dispondrá del periférico a dar de alta y un campo para introducir el número máximo de este periférico en el robot.

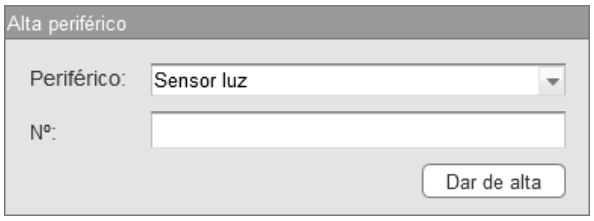

Ilustración 40: Ventana para dar de alta un periférico

- **Ventana para dar de baja un periférico::*Ventana baja periférico***

  Cuando el usuario selecciona la opción 'Dar de baja' un periférico, el sistema mostrará la ventana de la Ilustración 41, donde se puede observar que solo dispondrá del periférico a dar de baja.

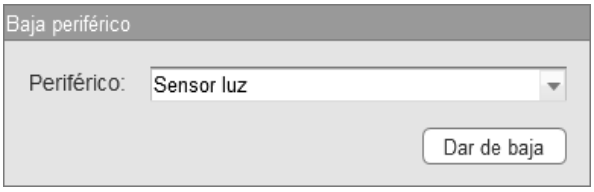

Ilustración 41: Ventana para dar de baja un periférico

- **Ventana para dar de alta una situación::*Ventana alta situación***

  Cuando la aplicación está en el modo avanzado, aparte de dar de alta y de baja los periféricos, el usuario podrá dar de alta o de baja las posibles situaciones en las que se puede en encontrar el robot y las acciones que podrá realizar.

  Cuando el usuario desee dar de alta una situación, el sistema mostrar la ventana que se ve en la Ilustración 42. En esta ventana se deberá seleccionar el periférico a cual se quiere añadir la situación, introducir la situación, introducir la función correspondiente a la situación indicada y finalmente seleccionar la opción Izquierda (I)/Derecha (D) si procede.

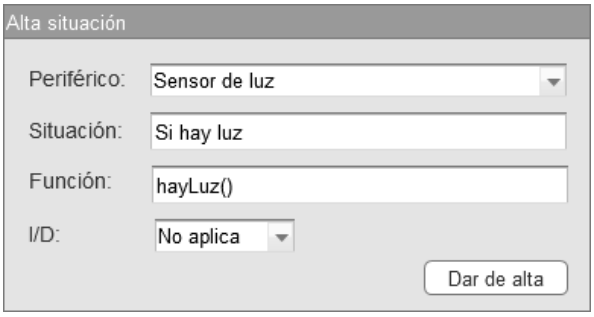

Ilustración 42: Ventana para dar de alta una situación

- **Ventana para dar de baja una situación::*Ventana baja situación***

  En el caso de que el usuario selecciona la opción 'Dar de baja' una situación, el sistema mostrará la ventana de la Ilustración 43, donde tendrá que seleccionar el periférico y la situación que se dará de baja.

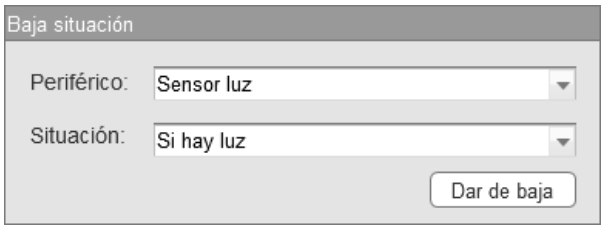

Ilustración 43: Ventana para dar de baja una situación

- **Ventana para dar de alta una acción::*Ventana alta acción***

  Tal y como se ha comentado anteriormente, el usuario también podrá dar de alta o de baja las acciones para cada periférico y las cuales el robot podrá realizar. Cuando el usuario seleccione la opción dar de alta el sistema abrirá la ventana que se muestra en la Ilustración 44, en la cual podrá seleccionar el periférico, introducir la acción e introducir la función que lleva a cabo la acción indicada.

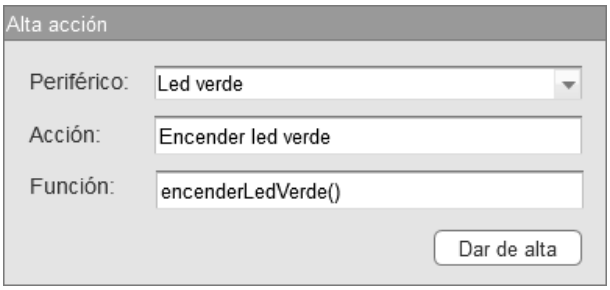

Ilustración 44: Ventana para dar de alta una acción

- **Ventana para dar de baja una acción::*Ventana baja acción***

  El usuario también podrá dar de baja una acción, para ello tendrá que seleccionar la opción 'Dar de baja' dando lugar a que el sistema muestre la

ventana de la Ilustración 45, en la cual el usuario podrá seleccionar el periférico y la acción que desea dar de baja.

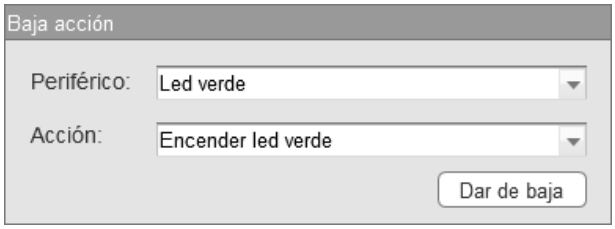

Ilustración 45: Ventana para dar de baja una acción

- **Ventana para seleccionar el puerto de comunicación::*Ventana seleccionar puerto***

  En cuanto al botón 'Cargar periféricos', al pulsar debe aparecer la ventana de la Ilustración 46, donde el usuario deberá seleccionar el puerto de comunicación con el robot.

Ilustración 46: Ventana para seleccionar el puerto de comunicación

- **Ventana con los periféricos cargados::*Ventana con los periféricos cargados***

  Cuando el usuario haya seleccionado el puerto de comunicación, automáticamente el sistema cargará los periféricos disponibles en el robot y actualizara los componentes de la interfaz gráfica tal y como se ve en la Ilustración 47. Esta lista de periféricos, como ya se ha visto en apartados anteriores los obtiene del fichero Perifericos.txt.

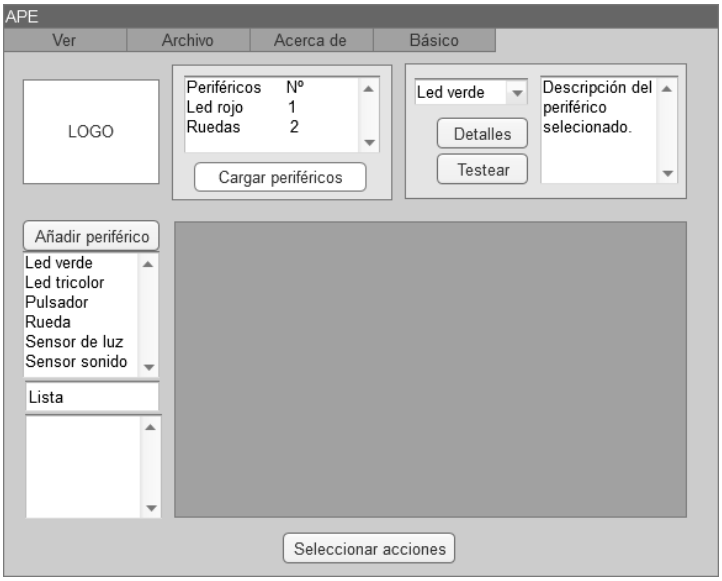

Ilustración 47: Ventana con los periféricos cargados

- **Ventana para testear los periféricos::*Ventana de test***

  En cuanto al botón 'Testear', al pulsar debe aparecer la ventana de la Ilustración 48, donde el usuario podrá realizar pruebas de cada uno de los periféricos disponibles en la aplicación.

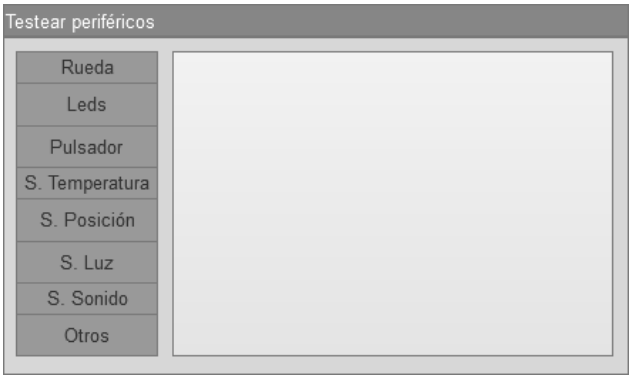

Ilustración 48: Ventana para testear los periféricos

- **Ventana para seleccionar las funcionalidades::*Ventana funcionalidades***

  Una vez elaborado el diseño en el entorno gráfico, el usuario podrá seleccionar las funcionalidades que desea que realice el robot. Estas funciones se cargarán dinámicamente dependiendo de los periféricos añadidos al diseño (ver Ilustración 49).

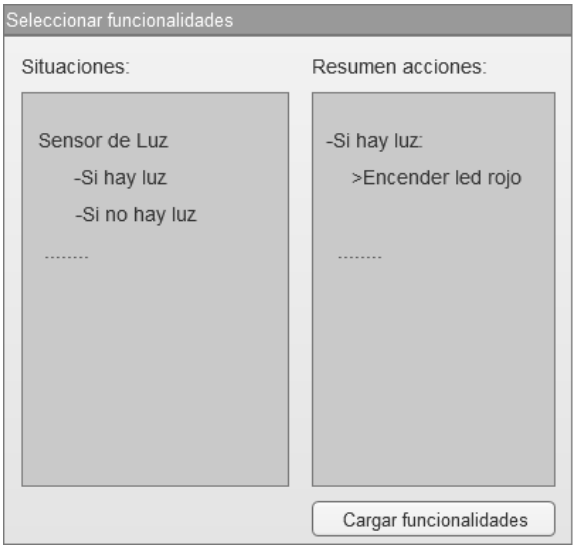

Ilustración 49: Ventana para seleccionar las funcionalidades

Esta ventana dispondrá de un botón para cargar las funcionalidades seleccionadas al robot y de un área de texto en la cual se mostrará el resumen de las funciones añadidas.

- **Ventana para seleccionar las acciones::*Ventana de acciones***

Esta ventana aparece cuando el usuario selecciona alguna de las situaciones indicadas en la ventana anterior. Del mismo modo, las opciones que apararen en esta ventana se cargan dinámicamente dependiendo de los periféricos disponibles en el diseño elaborado en el entorno gráfico.

Ilustración 50: Ventana para seleccionar las acciones

- **Ventana Acerca de..::*Ventana Acerca de..***

  Es la ventana de la Ilustración 51 que aparece cuando el usuario, desde la barra de menú superior, selecciona la opción 'Acerca de..'. En esta ventana se mostrará la versión e información de la aplicación.

Ilustración 51: Ventana Acerca de..

- **Diálogos de avisos a usuario::*Diálogos de aviso***

  Por último, la aplicación hará uso de varios diálogos de aviso en la cual informará al usuario de aspectos relacionados con las acciones que realice en la aplicación. En la Ilustración 52 se muestra el aspecto que tendrán estos diálogos.

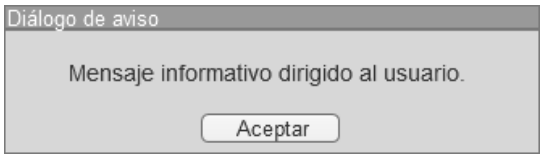

Ilustración 52: Diálogos de aviso

E Código funciones más relevantes

```
public class ViewDetailsButtonListener implements ActionListener{

    /*Atributos de la clase*/
    private final OpenAPEFrame parentFrame;

    /** ********************************************************************** ...6 lines */
    public ViewDetailsButtonListener(OpenAPEFrame frame)
    {...3 lines }

    /** ********************************************************************** ...6 lines */
    @Override
    public void actionPerformed(ActionEvent e)
    {...3 lines }
}
```

Ilustración 53: Interfaz ActionListener

```
/*****************************************************************************
 * Nombre: buttonDetailsPeripherals().
 * Descripción: Función que configura el botón para ver los detalles del
 *              periférico.
 * @return:
 *      -JButton boton: botón configurado.
 *****************************************************************************/
private JButton buttonDetailsPeripherals()
{
    JButton boton = new JButton("Ver detalles");
    boton.addActionListener(new ViewDetailsButtonListener(this));
    return boton;
}
```

Ilustración 54: Añadir evento a un botón

```
public class Sound implements Serializable
{
    /*Propiedades de la clase*/
    private final AudioClip click;
    private final URL urlClick;

    /** ********************************************************************** ...6 lines */
    public Sound(String sound)
    {...4 lines }

    /** ********************************************************************** ...4 lines */
    public void playSound()
    {...3 lines }

    /** ********************************************************************** ...4 lines */
    public void stopSound()
    {...3 lines }
}
```

Ilustración 55: Clase Sound.java

```
int main(void)
{
    SetupHardware();
    int i =0;
    //DDRD= _BV(REDLED) | _BV(GREENLED) ; //PINES LEDS COMO SALIDA OUTPUT
    int16_t byteRecibido;
    char buferDeComando[2];
    int indiceBuferComando=0;

    /* Create a regular character stream for the interface so that
    it can be used with the stdio.h functions */
    CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);

    LEDs_SetAllLEDs(LEDMASK_USB_NOTREADY);
    GlobalInterruptEnable();

    for (;;)
    {
        byteRecibido=-1;
        CheckJoystickMovement();

        // Revisar si se han recibido caracteres desde el Host
        while(CDC_Device_BytesReceived(&VirtualSerial_CDC_Interface)){
            //Recibir el nuevo byte
            byteRecibido = CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
            buferDeComando[indiceBuferComando++] = byteRecibido;
        }

        /*Los comandos van a ser de longitud 2*/
        if(indiceBuferComando == 2){
            procesarComando(buferDeComando, indiceBuferComando);
        }

        indiceBuferComando=0;
        CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
        USB_USBTask();
    }
}
```

Ilustración 56: Programa para el envío y recepción de datos

```
/*Abrimos el puerto USB para leer la lista de periféricos*/
CommunicationProtocol portCommunication = new CommunicationProtocol(port);
portCommunication.openCommunicationPort();
portCommunication.setParamsCommunicationPort();

/*Enviamos un comando para ver si es el puerto de comunicación COM es el correccto*/
portCommunication.writeBytesToPort("-C".getBytes());
while(flag)
{
    Thread.sleep(200);
    String conectado = portCommunication.readStringFromPort();
    if(conectado!=null && conectado.equals("Conectado")){
        serialPortName = port;
        break;
    }else
        flag = false;
}

if(serialPortName!= null){
    /*Cerramos el puerto de comunicación*/
    portCommunication.closePortCommunication();
    return true;
}

/*Cerramos el puerto de comunicación*/
portCommunication.closePortCommunication();
```

Ilustración 57: Función testPortCommunication

```
private int programMicroProtComunication()
{
    try {
        Runtime runtime = Runtime.getRuntime();
        Process proc = runtime.exec("bin/batchisp.exe -device atmega32u4 -hardware usb -operation erase"
                + " f memory flash blankcheck loadbuffer bin/virtualserial.hex program verify start reset 0");

        InputStream is = proc.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
            if(line.contains("Could not open USB device"))
                return 1;
            if(line.contains("Could not write to USB device"))
                return 2;
            if(line.contains("Cannot open file"))
                return 3;
        }
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(null,"Error en la comunicación con el dispositivo USB.", "Error de comun
    }
    return 0;
}
```

Ilustración 58: Función programMicroProtComunication

```
public int checkSerialPort()
{
    boolean flag = false;

    /*Recuperamos la lista de puertos localizados*/
    String[] lista = SerialPortList.getPortNames();

    /*Comprobamos si el puerto de comunicación es correcto*/
    for (String portName : lista)
        flag |= testPortComunication(portName);

    /*Programamos el micro el protocolo de comunicación VirtualSerial.hex entre pc y micro*/
    if((this.serialPortName == null))
        return programMicroProtComunication();

    /*Si el nombre del puerto es distinto de null --> Se muestra el estado de la comunicación*/
    if(this.serialPortName != null)
    {
        /*USB conectado en modo COM --> Mostrar ventana para eleccionar COM*/
        if(flag)
            return 0;

        /*No hay conexión al robot --> Programamos el micro el protocolo de comunicación entre pc y micro*/
        if(!flag)
            return programMicroProtComunication();
    }
    /*Todo ha ido bien --> muestra la ventana para seleccionar  puerto*/
    return 0;
}
```

Ilustración 59: Función checkSerialPort()

```
/*Contruimos el Programa main.c para posteriormente obtener el .hex y cargarlo al robot*/
createProgram();
/*Obtenemos el .hex*/
getFileHEX();
/*Comprrobamos si se ha generado nuestro fichero .hex*/
int rec =  checkFileHEX();
if(rec == 0)
    /*Aviso al usuario*/
    JOptionPane.showMessageDialog(null,"No se ha podido generar el fichero *.hex."+
            " Revise que las funciones añadidas esten correctamente definidas.","Fichero .he

/*Se ha creado el fichero .HEX correctamente*/
else if (rec == 1){
    int r = loadHEXToMicro();
    /*Programamos el .hex al micro*/
    switch(r)
    {
        case 0:
```

Ilustración 60: Cargar programa al robot

Rutinas.h

```
68  /*Nuevas funciones SITUACIONES*/
69  bool hayContactoIzquierdo();
70  bool hayContactoDerecho();
71  bool hayObjetoA(int distancia);
72  bool hayObjeto();
73  bool hayLuz();
74  bool temperatura(int temperatura);
75  bool temperaturaEntre(int tempIni, int tempFin);
76  bool seguidorLineaBlancaIzquierdo();
77  bool seguidorLineaNegraIzquierdo();
78  bool seguidorLineaBlancaIDerecho();
79  bool seguidorLineaNegraDerecho();
80  bool pulsador1Presionado();
81  bool pulsador2Presionado();
82
83  /*Nuevas funciones ACCIONES*/
84  void zumbidoBajo();
85  void zumbidoMedio();
86  void zumbidoAlto();
87  void mostrarMensaje();
88  void encenderLedVerde();
89  void apagarLedVerde();
90  void encenderLedRojo();
91  void apagarLedRojo();
92  void encenderTricolorLedAzul();
93  void apagarTricolorLedAzul();
94  void encenderTricolorLedRojo();
95  void apagarTricolorLedRojo();
96  void encenderTricolorLedVerde();
97  void apagarTricolorLedVerde();
98  void avanzarRobot();
99  void retrocederRobot();
100 void girarIzquierdaRobot();
101 void girarDerechaRobot();
```

Ilustración 61: Rutinas.h

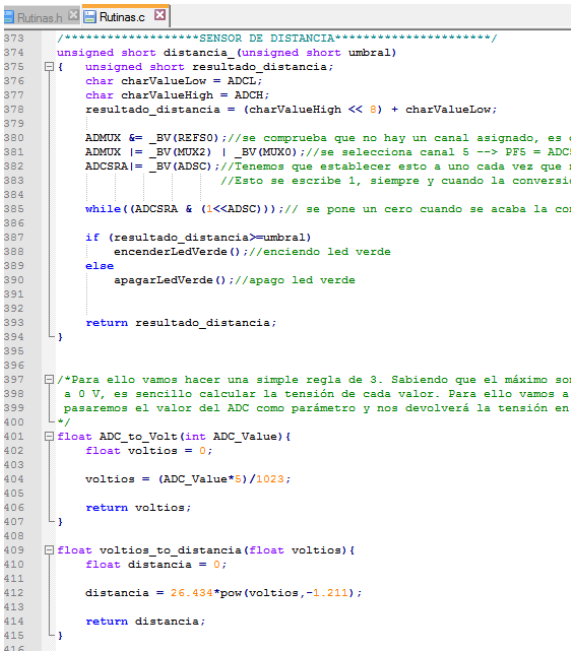

Ilustración 62: Rutinas.c

F Manual de Programación USB DFU Bootloader

Programar USB DFU Bootloader mediante el programador AVRISP mkII

1- Se deberá abrir la herramienta Atmel Studio 6.0.

2- Una vez dentro de la herramienta se debe seleccionar el botón "Device programming", y se abrirá la pantalla como la mostrada en la ilustración 63.

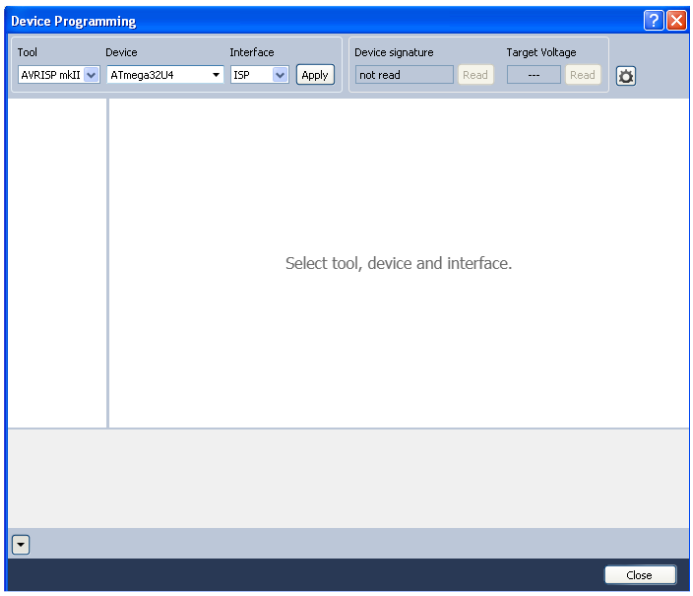

Ilustración 63: Pantalla "Device programming"

Se debe seleccionar en Tool la opción "AVRISPmkII000200015261", en Device el microcontrolador Atmega32u4 y la Interfaz ISP. Cuando este todo seleccionado se debe pulsa el botón "Apply".

3- Ahora se puede ver un desglose de opciones a la izquierda, antes de continuar se debe configurar el reloj del ISP, debe ser más baja que ¼ de la frecuencia de funcionamiento, se recomienda usar 250kHz aunque se puede subir hasta 2MHz.

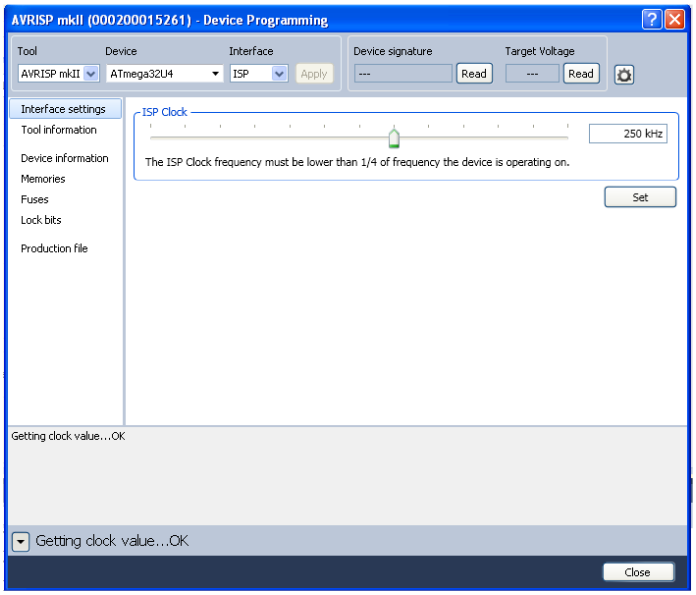

Ilustración 64: AVRISPmkII000200015261

4- Una vez definido el reloj, ahora en la pestaña de “Memories”, en el apartado “Flash” se debe subir el bootloader. Una vez subido se pulsa “Program”.

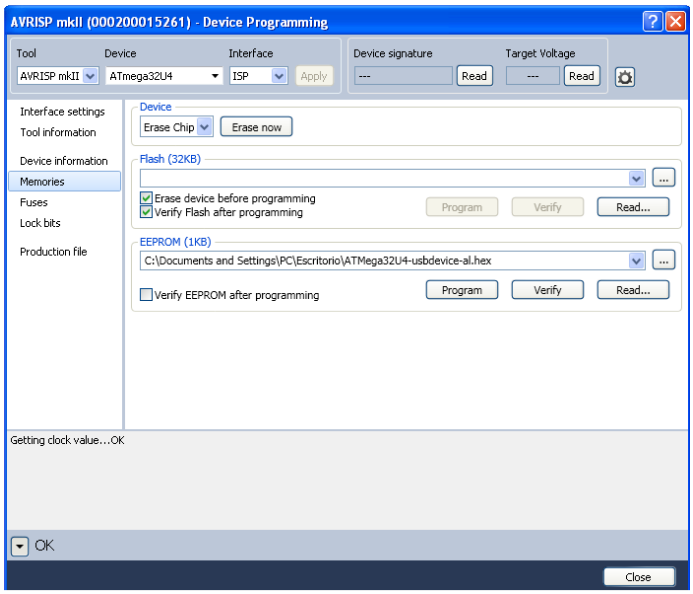

Ilustración 65: Cargar bootloader

5- Tras haber cargado el bootloader en el microcontrolador, se conecta el robot al ordenador por el MiniUSB.

6- Se abre el administrador de dispositivos que se encuentra en el "Panel de control, Sistema y seguridad, sistema” y se puede observar el dispositivo ATm32U4DFU con un icono amarillo lo cual indica que el dispositivo DFU no está instalado correctamente.

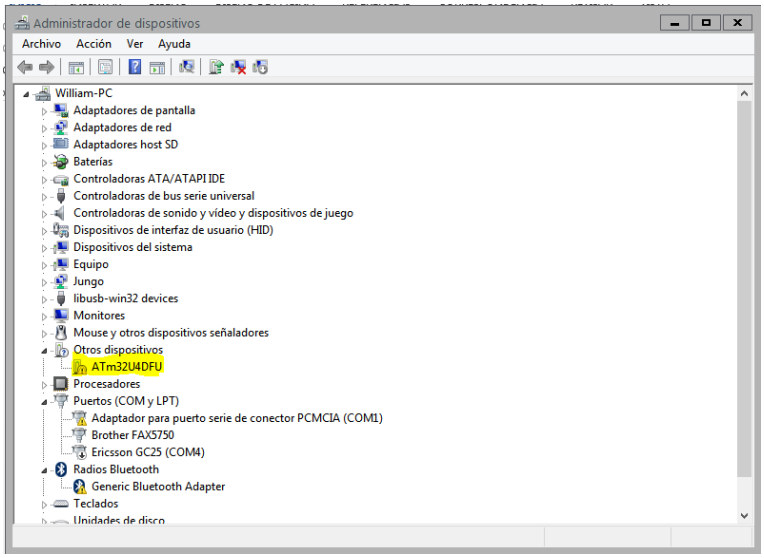

Ilustración 66: Dispositivo ATm32U4DFU

7- Se selecciona "Actualizar Software de controlador.. " (ver Ilustración 67) y posteriormente se elige "Buscar software de controlador en el equipo" (ver Ilustración 68).

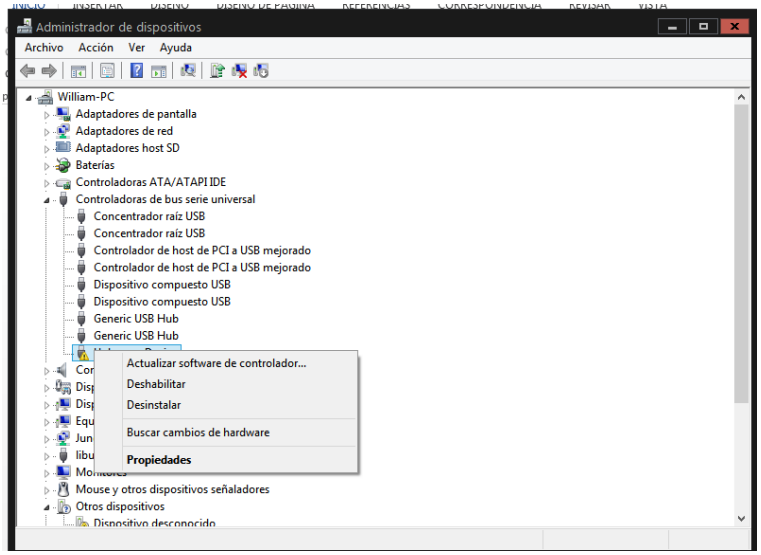

Ilustración 67: Actualizar software de controlador

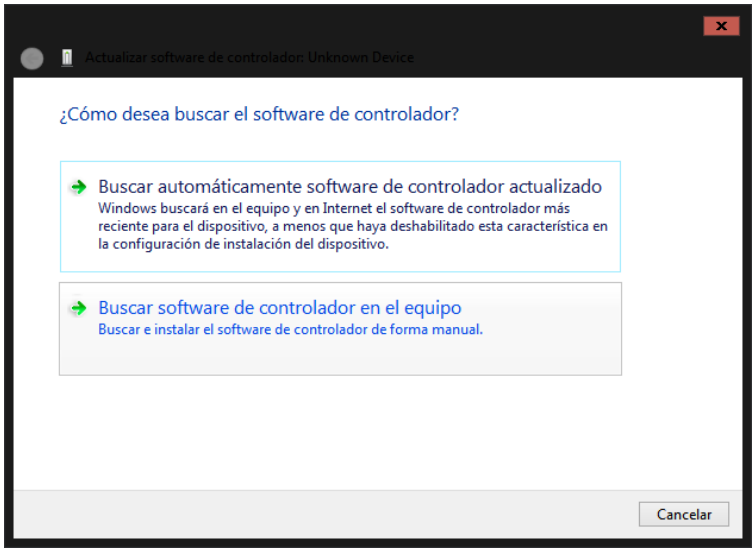

Ilustración 68: Buscar software de controlador en el equipo

8- Nos dirigiremos a la carpeta de instalación FLIP USB ("C:\Archivos de programa\Atmel\Flip 3.x.x\USB").

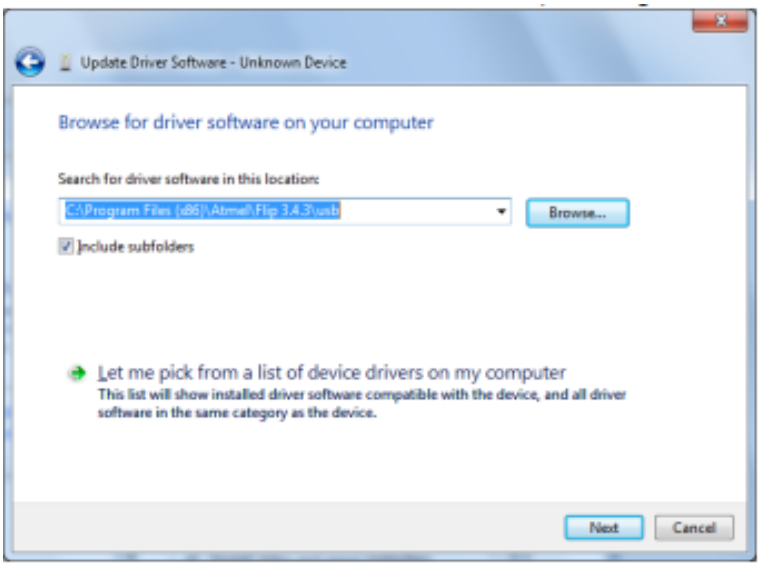

Ilustración 69: Seleccionar carpeta de instalación de FLIP

9- En los sistemas operativos de 64 bits, es posible que aparezca esta advertencia.

Ilustración 70: Advertencia Sistemas operativos de 64 bits

10- Seleccione la opción "Instalar este software de controlador de todas formas".

11- Tras la instalación en el Administrador de Dispositivos se pueder ver el dispositivo ATmega32U4.

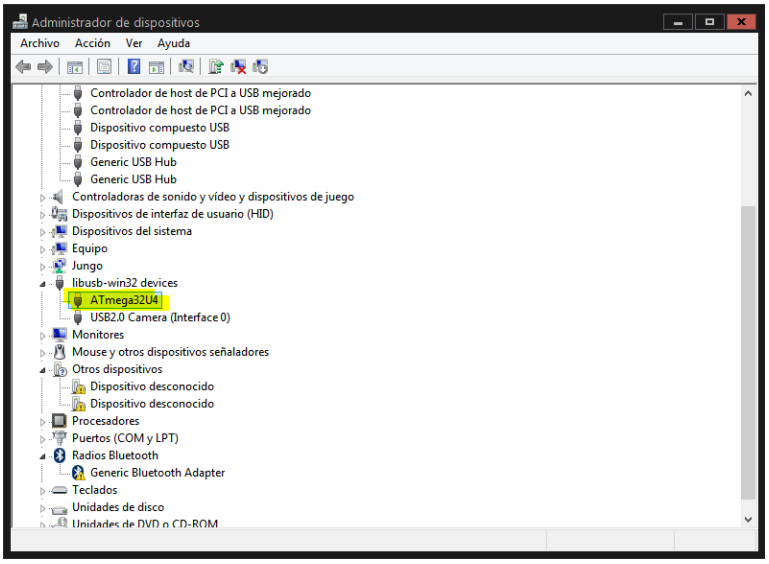

Ilustración 71: Dispositivo ATmega32U4

G Manual instalación controlador LUFA CDC Demo

Cuando se conecte el robot al pc a través del USB en el Administrador de Dispositivos aparecerá el Dispositivo LUFA CDC Demo.

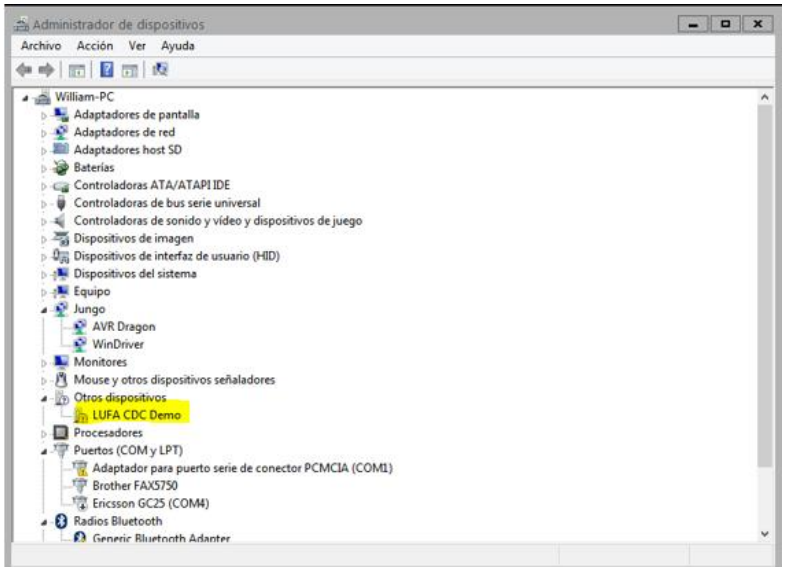

Ilustración 72: LUFA CDC Demo

1- Se deberá hacer clic derecho sobre el dispositivo LUFA CDC Demo → Actualizar Software de Controlador y aparecerá la ventana de Ilustración 74.

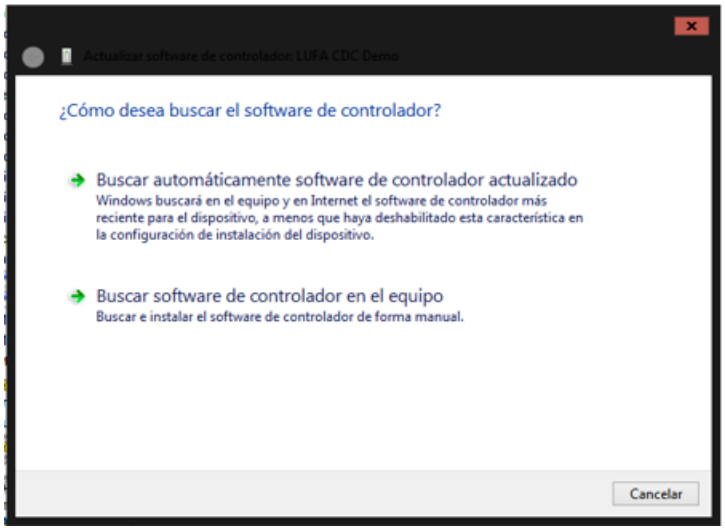

Ilustración 73: Actualizar software de controlador

2- Se seleccionará la opción “Buscar software de controlador en el equipo” (Ilustración 74).

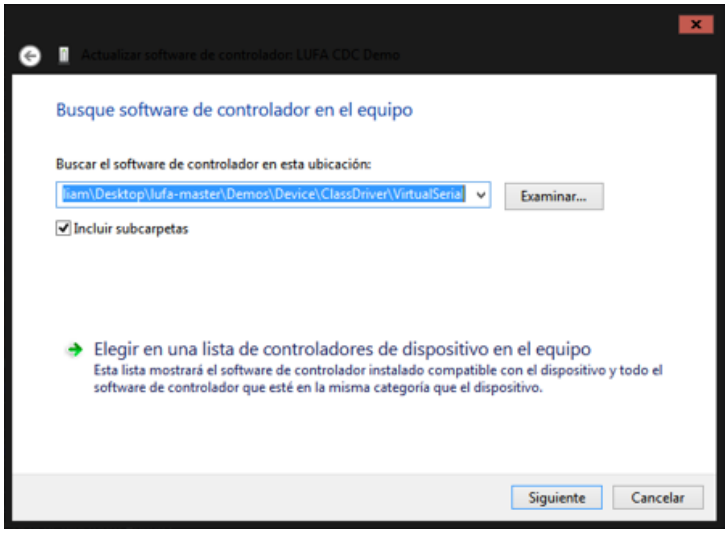

Ilustración 74: Buscar software de controlador en el equipo

3- Se añadirá la ruta donde se encuentra el fichero LUFA VirtualSerial.ini

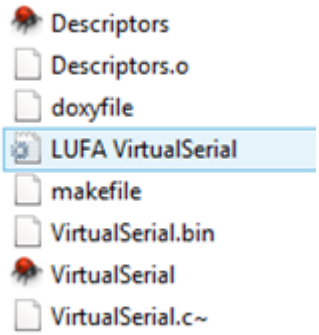

Ilustración 75: Fichero VirtualSerial.ini

4- En los sistemas operativos de 64 bits, es posible que aparezca esta advertencia.

Ilustración 76: Advertencia Sistemas operativos de 64 bits

5- Seleccione la opción "Instalar este software de controlador de todas formas".

6- Tras la instalación, en el Administrador de Dispositivos se puede ver el dispositivo ***LUFA CDC-ACM Virtual Serial Port (COM8)*** listo para intercambiar información entre el PC y robot.

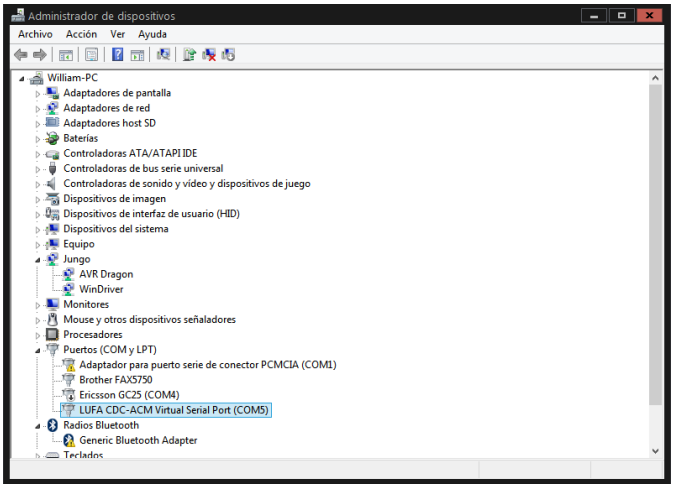

Ilustración 77: LUFA CDC-ACM Virtual Serial Port (COM8)

H Interfaz de usuario

H.1 Ventana de inicio

Al ejecutar la aplicación nos va a salir la pantalla de la Ilustración 78 que inicialmente no tienen cargado ningún periférico ni datos en la interfaz gráfica. Pero el usuario sí que podrá cargar los periféricos pulsando sobre el botón "Cargar periféricos" y dar de alta o baja periféricos en la aplicación. Solo puede realizar esta última acción ya que la pantalla que aparece inicialmente está en el modo de dificultad "Básico".

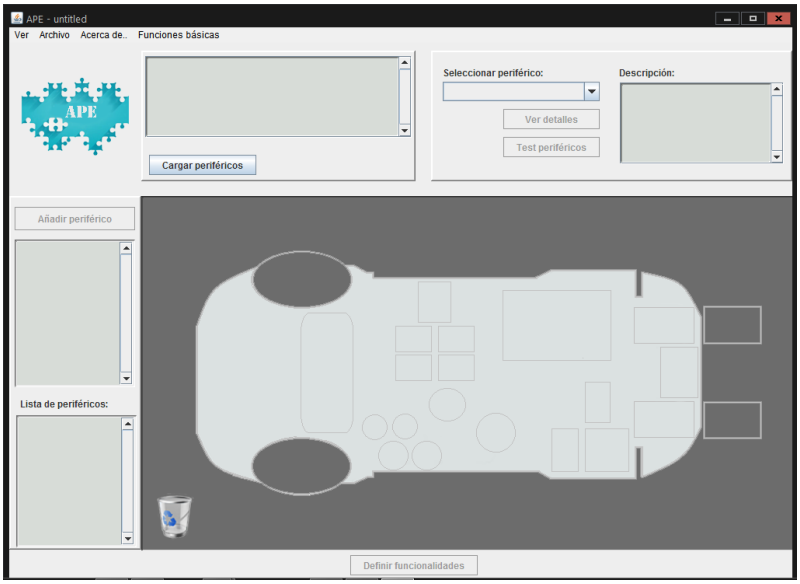

Ilustración 78: Pantalla de inicio

H.2 Ventana para seleccionar el puerto de comunicación

Cuando el usuario pulse sobre el botón "Cargar periféricos" el sistema instalará en el robot el software para poder establecer conexión entre el robot y el pc. Tras ello la aplicación tendrá que comprobar si hay algún puerto de comunicación conectado al pc, y en caso de que existiese, el sistema enviará un comando a través del puerto USB para comprobar que el puerto es el correcto. Si la aplicación no recibe ninguna respuesta a través del puerto USB el sistema descartará dicho puerto y seguirá comprobando los puertos que estén conectados al pc. Si no hay ningún puerto de comunicación conectado al pc o si los hubiese pero no corresponden con el puerto de comunicación, la aplicación notificará de ello con el siguiente mensaje indicándonos posibles soluciones (Ilustración 79).

Ilustración 79: Aviso de Error al establecer conexión

En cuanto la aplicación reciba una respuesta a través del puerto USB, el sistema mostrará la ventana de la Ilustración 80 donde se podrá seleccionar el puerto de comunicación correspondiente. Esta ventana tendrá asociada dos botones: Aceptar y Cancelar.

Ilustración 80: Ventana para seleccionar el puerto

H.3 Ventana con los periféricos cargados

Una vez se haya seleccionado el puerto de comunicación, el usuario podrá pulsar sobre el botón "Aceptar" dando lugar a que la pantalla principal cargue de forma automática los periféricos presentes en el robot (ver Ilustración 81). Estos periféricos están almacenados en el fichero Perifericos.txt al cual el usuario puede añadir o eliminar periféricos desde las ventanas "Dar de alta periférico" y "Dar de baja periférico" respectivamente.

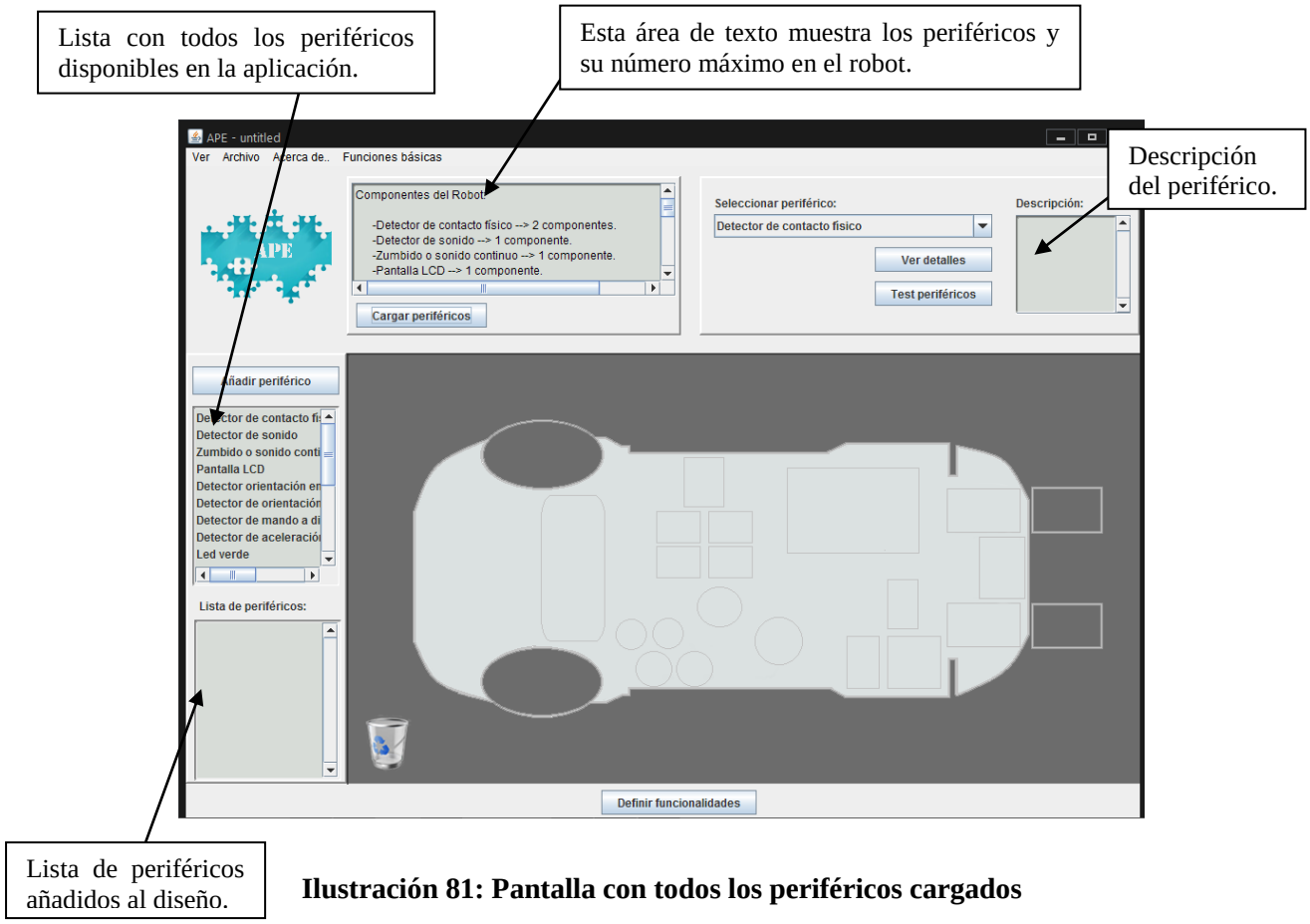

Ilustración 81: Pantalla con todos los periféricos cargados

Si el usuario pulsa el botón “Cancelar” se cerrará la ventana sin actualizar ningún componente de la interfaz gráfica.

Una vez se esté en la ventana de la Ilustración 82 el usuario podrá hacer uso del resto de funcionalidades que dispone la interfaz gráfica. Entre ellas está el botón “Añadir periférico” que añadirá un periférico en el entono gráfico siempre y cuando se haya seleccionado previamente. En el caso de que el usuario no haya seleccionado un periférico el sistema mostrará el mensaje de advertencia de la Ilustración 83.

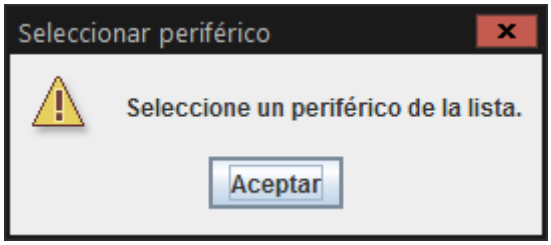

Ilustración 82: Mensaje de advertencia

Si no hay problemas al añadir el periférico al entorno gráfico, el periférico se añadirá a la parte superior tal y como se puede observar en la Ilustración 83.

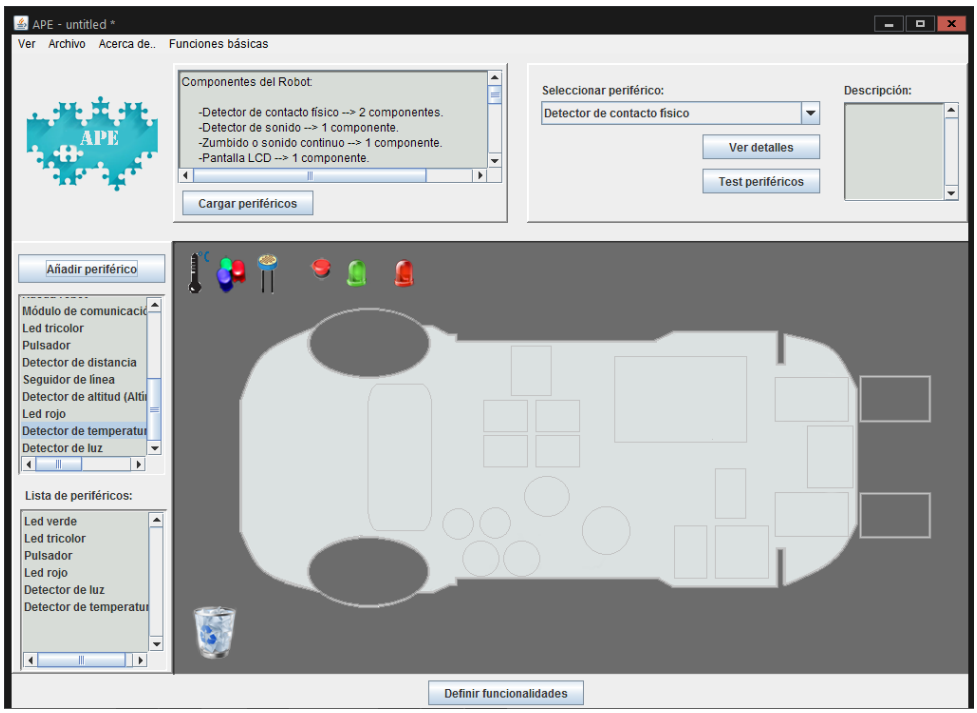

Ilustración 83: Añadir periférico

Cada periférico añadido en el entorno gráfico podrá ser eliminado de forma individual, arrastrándolo a la papelera que se encuentra en la esquina inferior izquierda o haciendo clic sobre el periférico y seleccionándola opción “Eliminar”. También es posible eliminar todos los periféricos disponibles en el entorno gráfico, seleccionado la opción “Eliminar todos” (ver Ilustración 84).

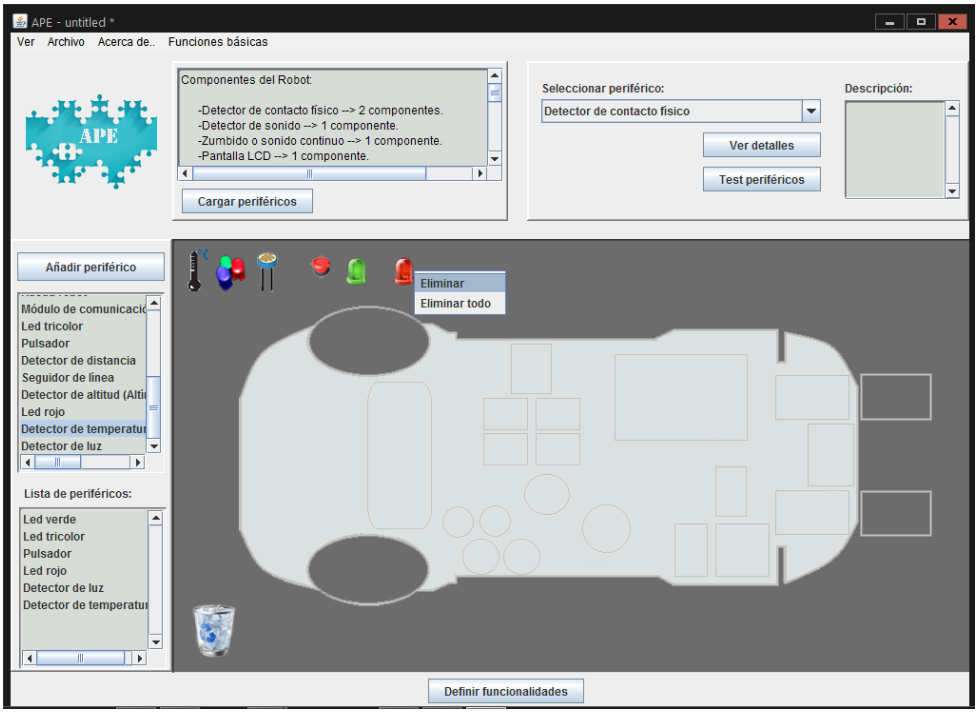

Ilustración 84: Eliminar/Eliminar todo periférico

Como ya se ha observado, en el entorno gráfico existe un diseño al cual se irán añadiendo cada uno de los periféricos según la necesidad del usuario. Este diseño estará dotado de varias formas rectangulares que hacen referencia a los sensores y formas redondas/ovaladas que hacen referencia a los actuadores que tendrá el robot. Cada vez que el usuario pase el cursor del ratón sobre una de estas formas, el sistema mostrará un pequeño mensaje indicando el periférico correspondiente a esa posición (ver Ilustración 85).

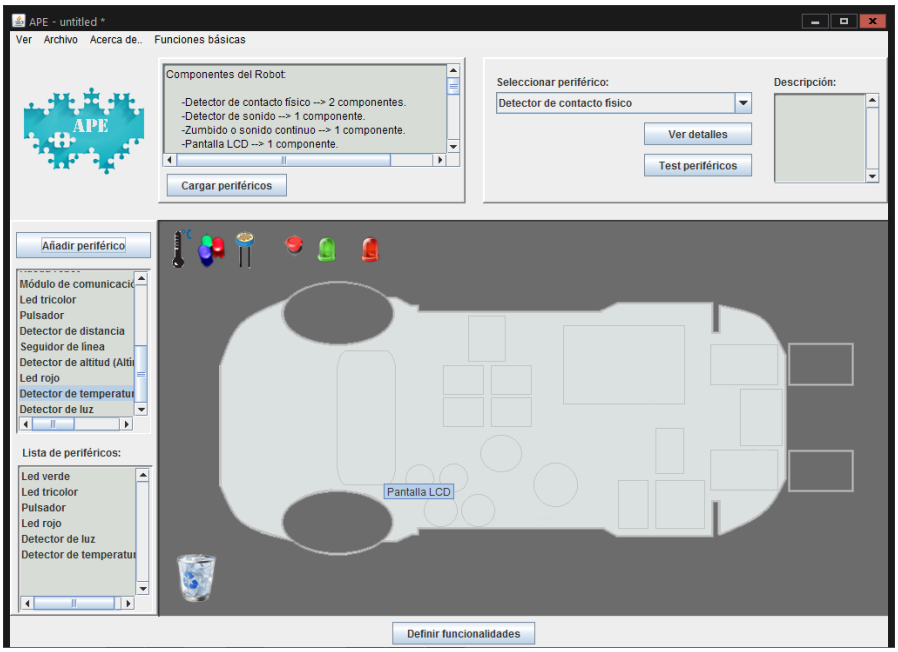

Ilustración 85: Nombre del periférico

Por otro lado, se puede ver que existe el botón "Ver detalles" cuya finalidad es mostrar una pequeña descripción del periférico seleccionado (ver ilustración 86), si el usuario no ha seleccionado ningún periférico de la lista, el sistema mostrará el mensaje de advertencia de la ilustración 87.

Ilustración 86: Detalles del periférico

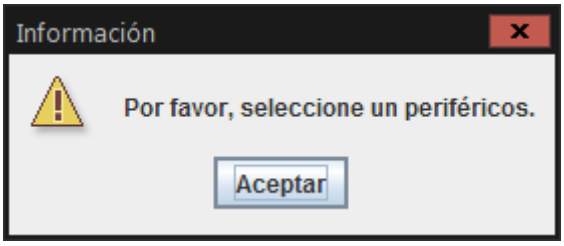

Ilustración 87: Mensaje de Advertencia

H.4 Ventana Test periféricos

El usuario también tendrá la posibilidad de poder comprobar el funcionamiento de cada uno de los periféricos presentes en la aplicación, esto lo realizará pulsando sobre el botón "Test periféricos". Si no hay ningún puerto de comunicación conectado al pc o si los hubiese pero no corresponden con el puerto de comunicación, la aplicación notificará de ello con el siguiente mensaje indicándonos posibles soluciones.

Ilustración 88: Error al establecer conexión

Si no hay problemas al pulsar sobre el botón "Test periféricos", el sistema abrirá la ventana para testear cada uno de los periféricos disponibles en la aplicación (ver Ilustración 89).

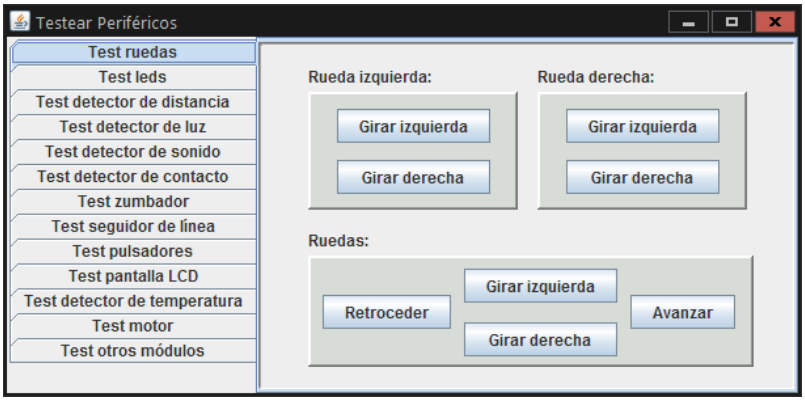

Ilustración 89: Ventana Test periféricos

(1) Test ruedas

En esta pestaña se comprobará el correcto funcionamiento de las ruedas que dispone el robot probándolas tanto individualmente como conjuntamente.

(2) Test leds

En esta pestaña se comprobará el correcto funcionamiento de los leds, tanto del led verde, rojo y tricolor. Simplemente se indicará al robot, con el comando correspondiente, que encienda o apague uno de estos leds.

(3) Test detector de distancia

Cuando el usuario acceda a esta pestaña, el sistema enviará un comando al robot y se quedará escuchando el puerto de comunicación pendiente de recibir la información procedente del sensor de proximidad. Una vez la aplicación reciba la información proporcionada por este sensor el sistema actualizará la distancia que ha detectado.

(4) Test detector de luz

Cuando el usuario acceda a esta pestaña, el sistema enviará un comando al robot y se quedará escuchando el puerto de comunicación pendiente de recibir la información que enviará el sensor de luz. Una vez la aplicación reciba la información proporcionada por este sensor el sistema actualizará el umbral de luz que incide sobre este sensor.

(5) Test detector de sonido

El sistema enviará un comando al robot y se quedará escuchando el puerto de comunicación pendiente de recibir la información que enviará el sensor de sonido. Una vez la aplicación reciba la información proporcionada por este sensor el sistema actualizará el umbral de sonido que incide sobre este sensor.

(6) Test detector de contacto físico

Cuando se acceda a esta pestaña, la aplicación enviará un comando al robot y se quedará escuchando el puerto de comunicación pendiente de recibir la información procedente del sensor de contacto físico. Una vez la aplicación reciba la información proporcionada por este sensor el sistema actualizará el texto y una imagen dependiendo de si hay contacto o no.

(7) Test zumbador

El usuario podrá comprobar el correcto funcionamiento de este periférico, pulsando sobre los botones disponibles en esta pestaña. Dependiendo del botón pulsado el sistema enviara un comando u otro para que le robot pueda interpretar y saber que acción llevar a cabo. Dependiendo del botón pulsado este periférico emitirá un sonido con más o menos frecuencia.

(8) Test seguidor de línea

El sistema enviará un comando al robot y se quedará escuchando el puerto de comunicación pendiente de recibir la información que enviará este periférico. Una vez la aplicación reciba la información proporcionada por este sensor el sistema actualizará una imagen dependiendo del color de línea detectado.

(9) Test pulsadores

El sistema enviará un comando al robot y se quedará escuchando el puerto de comunicación pendiente de recibir la información que enviará tras pulsar uno de los botones (de propósito general) disponibles en el robot. Una vez el usuario haya pulsado uno de estos botones el robot enviará un comando a la interfaz indicando el botón que se ha pulsado dando lugar a que el sistema actualice el texto y la imagen correspondiente al botón pulsado.

(10) Pantalla LCD

En este apartado el usuario introducirá un dato en un campo de texto y enviará dicho dato al robot, el cual deberá saber interpretarlo y mostrarlo en la pantalla LCD.

(11) Test detector de temperatura

Cuando el usuario acceda a esta pestaña, el sistema enviará un comando al robot y se quedará escuchando el puerto de comunicación pendiente de recibir la información procedente del sensor de temperatura. Una vez la aplicación reciba la información proporcionada por este sensor el sistema actualizará la temperatura que incide sobre este sensor.

(12) Test otros modulos

En esta pestaña se podrá testear el giroscopio, acelerómetro, altímetro y la brújula. En todos estos sensores, la aplicación mostrará en un campo de texto la información captada por cada uno de estos periféricos.

H.5 Ventana Definir funcionalidades

Una vez el usuario haya añadido varios periféricos al diseño expuesto en el entorno de trabajo (ver Ilustración 90), podrá seleccionar las funciones que desea que realice el robot.

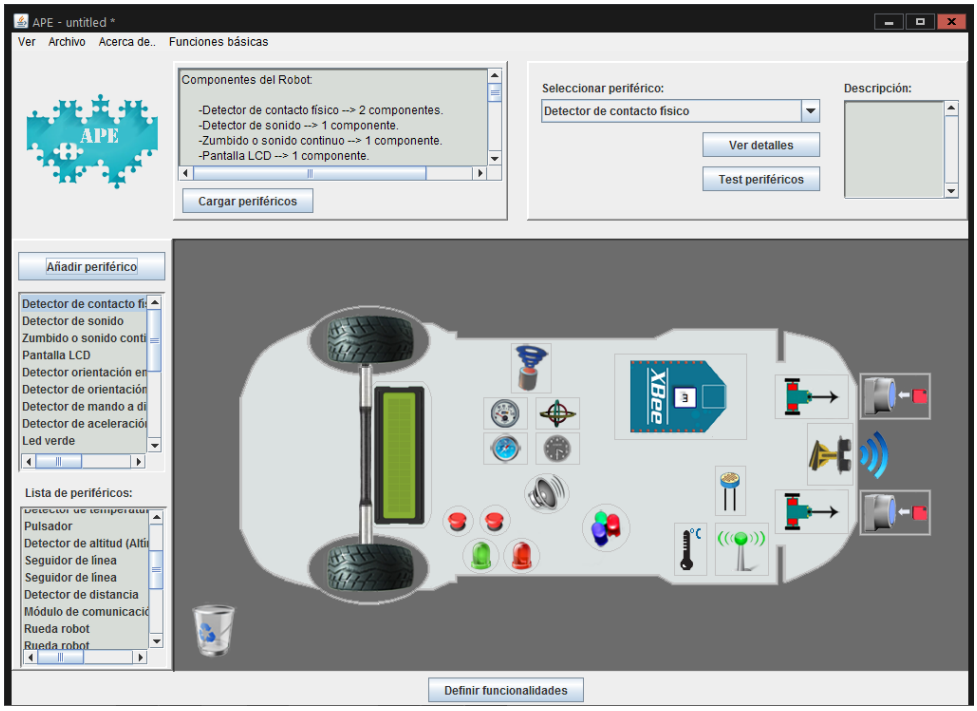

Ilustración 90: Periféricos en el diseño

Para ello tendrá que pulsar sobre el botón “Definir funcionalidades” que abrirá la ventana donde se definen todas las funcionalidades del robot. Antes de abrir esta ventana el sistema comprobará que el diseñó cumple con ciertas condiciones para poder definir las funcionalidades deseadas. Por ejemplo, si el usuario pulsa sobre el botón “Definir funcionalidades” y no hay periféricos añadidos al diseño entonces el sistema mostrará el mensaje de aviso de la Ilustración 91.

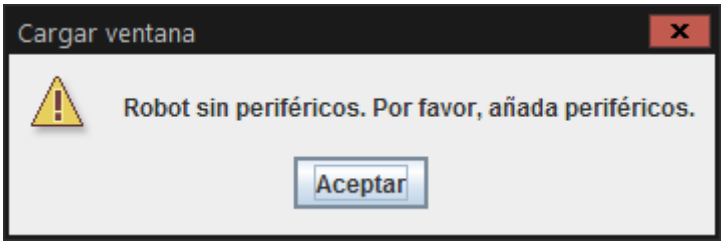

Ilustración 91: Mensaje de aviso

También puede ocurrir que el usuario haya añadido simplemente las ruedas del robot y ningún otro periférico, en este caso el sistema informará al usuario indicándole los posibles periféricos que puede añadir al diseño (ver Ilustración 92).

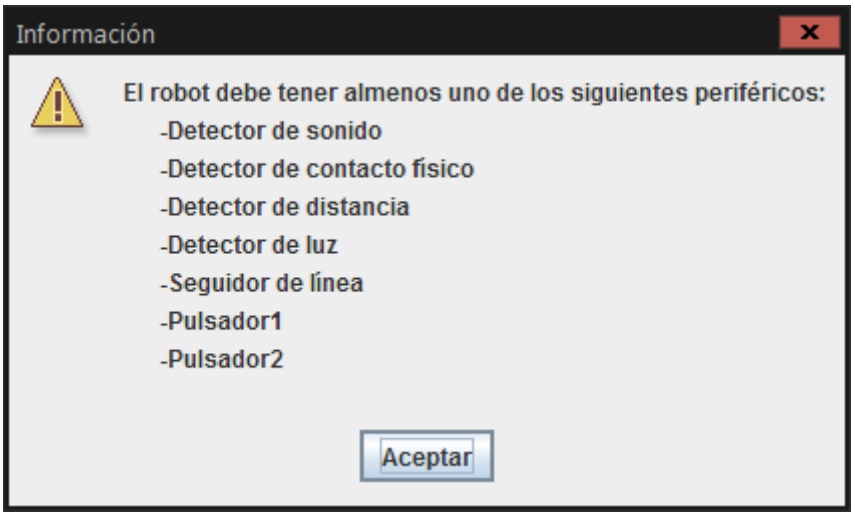

Ilustración 92: Posibles periféricos a añadir

Si el diseño tiene varios periférico pero no dispone de las ruedas del robot, no tiene sentido que el sistema de la posibilidad de definir las funcionalidades, con lo cual, mostrará un mensaje de advertencia indicando al usuario que añada las ruedas del robot al diseño (ver Ilustración 93).

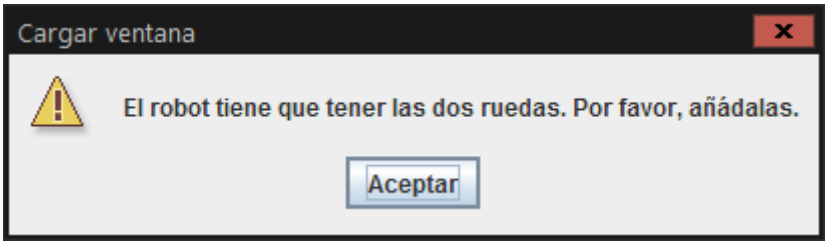

Ilustración 93: Añadir ruedas al robot

En caso de que todo este correcto, la aplicación mostrará la ventana para seleccionar las funcionalidades que llevará a cabo el robot (ver Ilustración 95). En esta nueva ventana se visualizará las situaciones en la que se puede encontrar el robot, que serán cargadas dinámica dependiendo de los periféricos añadidos al diseño. Estas situaciones están almacenadas en el fichero Situaciones.txt al cual el usuario puede añadir o eliminar desde las ventanas "Dar de alta situación" o "Dar de baja situación" respectivamente.

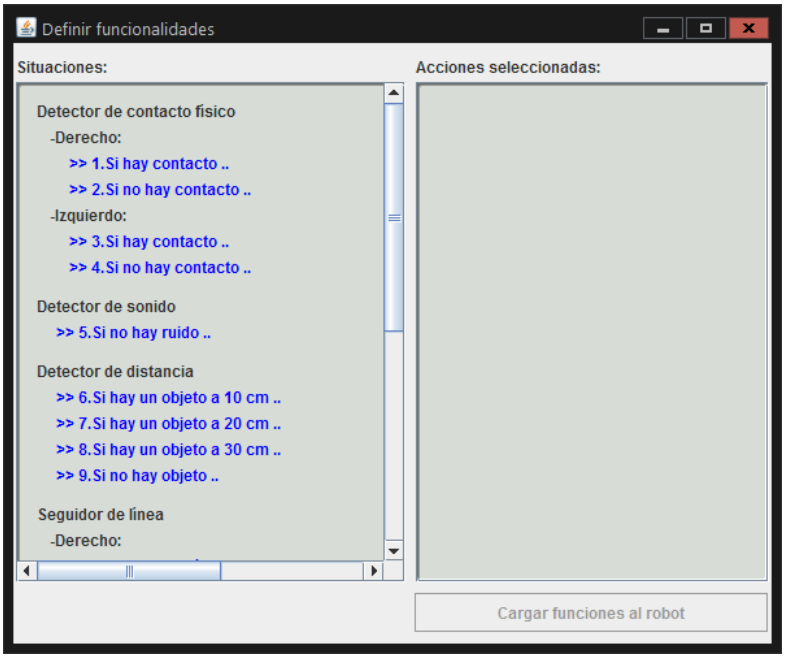

Ilustración 94: Ventana para definir las funcionalidades del robot

H.6 Ventana Seleccionar acciones

Cada vez que el usuario seleccione una **situación**, el sistema mostrará las **acciones** que puede llevar acabo cada periférico del robot (ver Ilustración 95). Del mismo modo estas acciones serán cargadas de forma dinámica dependiendo de los periféricos añadidos al diseño. Estas acciones están almacenadas en el fichero Acciones.txt al cual el usuario puede añadir o eliminar desde las ventanas "Dar de alta acción" o "Dar de baja acción" respectivamente.

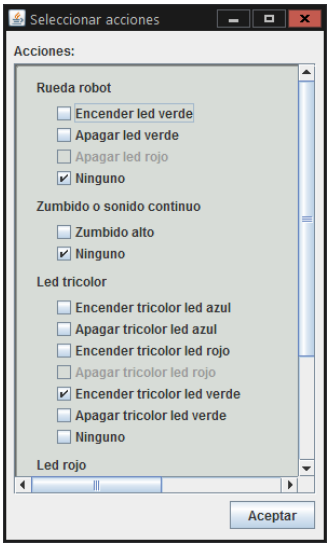

Ilustración 95: Ventana para seleccionar las acciones

Una vez se haya seleccionado las acciones que llevará a cabo cada periférico del robot, si el usuario pulsara sobre el botón "Aceptar" abandonará esta ventana y automáticamente en la "Ventana Definir funcionalidades" se actualizará el panel de la derecha con el resumen de las acciones que ha ido seleccionando el usuario para cada situación (ver Ilustración 96).

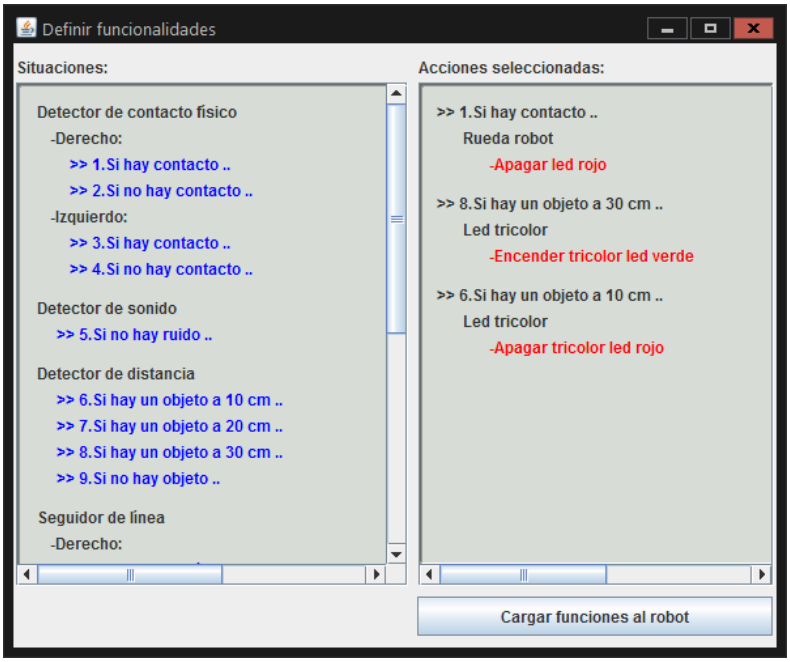

Ilustración 96: Ventana con el panel resumen actualizado

Cuando el usuario haya definido todas las funciones que desea que realice el robot, éste podrá cargarlo en el robot pulsando sobre el botón "Cargar funciones al robot". El sistema en este instante comprobará que exista conexión con el robot a través del puerto de comunicación, y si es así programará el robot con todas las funcionalidades definidas. Si la aplicación no logra establecer ningún tipo de

comunicación con el robot el sistema informará de ello al usuario mediante el mensaje de advertencia de la Ilustración 97.

Ilustración 97: Error al cargar el programa

H.7 Ventana Alta periféricos

Esta ventana se abre desde la barra de menú situada en la parte superior de la aplicación, cuando el usuario selecciona la opción para dar de alta un periférico.

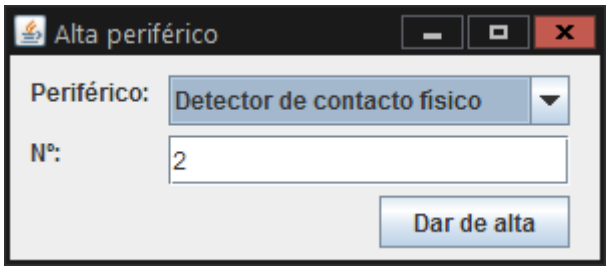

Ilustración 98: Alta periférico

Si el periférico existe en el fichero Periféricos.txt el sistema mostrará el aviso de la Ilustración 100 informando al usuario de ello.

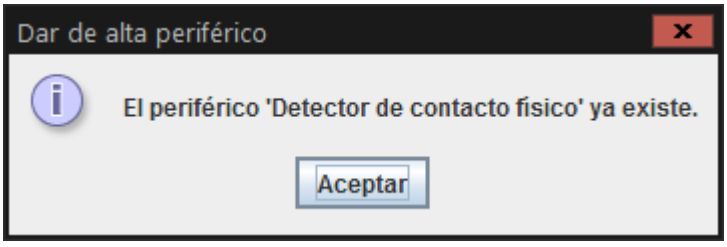

Ilustración 99: Mensaje de información

En el caso de que el periférico no exista en este fichero, el sistema lo añadirá y mostrara el siguiente mensaje.

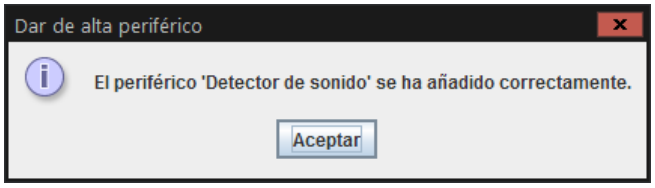

Ilustración 100: Mensaje Alta periférico

H.8 Ventana Baja periféricos

La ventana que se observa en la Ilustración 101 corresponde a la ventana en la que se da de baja un periférico.

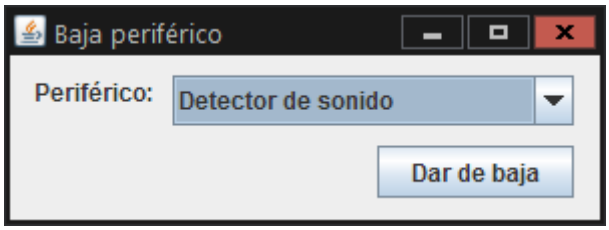

Ilustración 101: Baja periférico

En el instante en el que el usuario pulsa sobre el botón "Dar de baja" el sistema eliminará del fichero Perifericos.txt el periférico seleccionado e informará de ello al usuario.

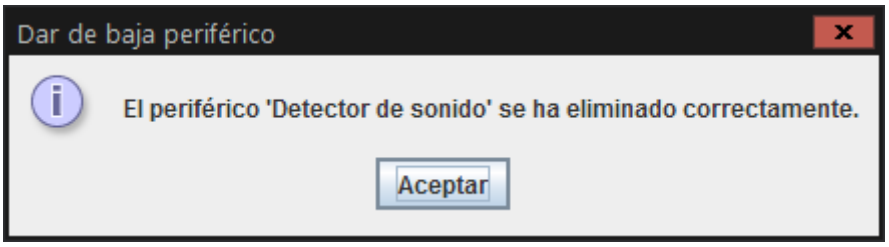

Ilustración 102: Mensaje Baja periférico

H.9 Ventana Alta situación

En la ventana de la Ilustración 103 se puede observar los campos necesarios para dar de alta una situación. En primer lugar el usuario tendrá que seleccionar el periférico, posteriormente introducirá la situación, la función correspondiente a esta situación y finalmente seleccionará el campo I/D si procede.

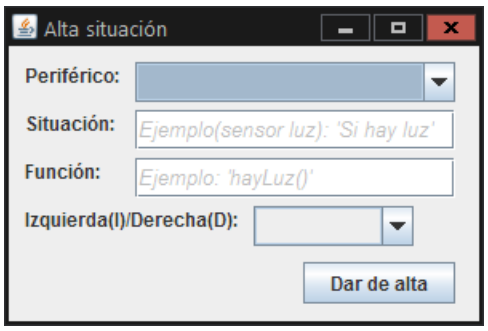

Ilustración 103: Alta situación

Cada vez que el usuario pulse sobre el botón “Dar de alta” el sistema comprobará que los campos estén rellenos, en caso de que no lo estén el sistema mostrará un mensaje de aviso indicando al usuario que debe rellenar dichos campos.

Si todo ha ido correctamente, el sistema indicaría al usuario que la situación se ha dado de alta correctamente tal y como se puede ver en la Ilustración 104.

Ilustración 104: Mensaje Alta situación

Puede ocurrir que la situación que se va a dar de alta ya exista en el fichero Situaciones.txt, en este caso el sistema informará de ello con el mensaje siguiente.

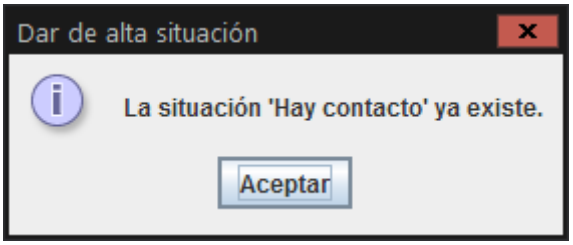

Ilustración 105: Mensaje de información

H.10 Ventana Baja situación

La ventana que se observa en la Ilustración 106 corresponde a la ventana en la que se da de baja una situación correspondiente a un periférico. Cada vez que seleccionemos el periférico se cargaran automáticamente las situaciones correspondientes a dicho periférico en el campo “Situación”.

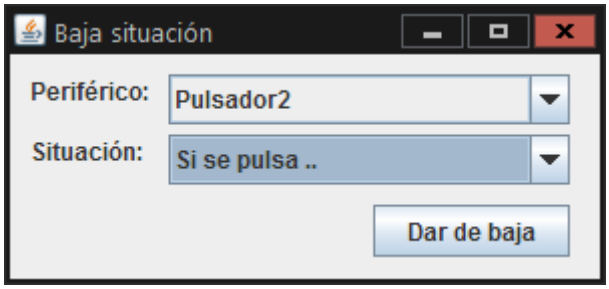

Ilustración 106: Baja situación

En el instante en el que el usuario pulsa sobre el botón “Dar de baja” el sistema eliminará del fichero Situaciones.txt la línea correspondiente al periférico y la situación e informará de ello al usuario.

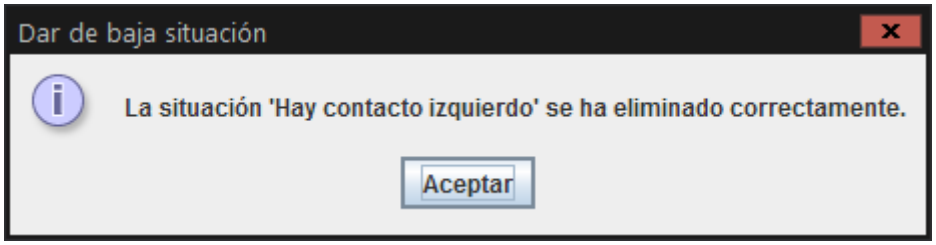

Ilustración 107: Mensaje Baja situación

H.11 Ventana Alta acción

En la ventana de la Ilustración 108 se puede observar los campos necesarios para dar de alta una acción para un periférico. En primer lugar el usuario tendrá que seleccionar el periférico, posteriormente introducirá la acción que realizará este periférico y finalmente la función correspondiente a esta acción.

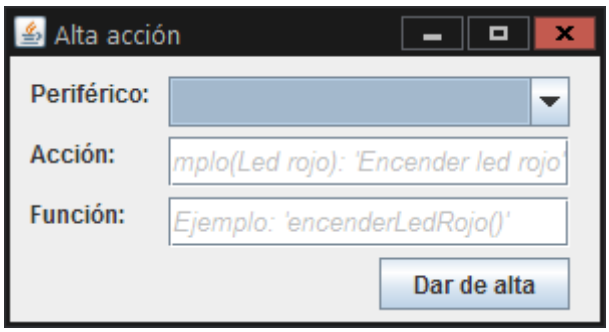

Ilustración 108: Alta acción

Cada vez que el usuario pulse sobre el botón “Dar de alta” el sistema comprobará que los campos estén rellenos, en caso de que no lo estén el sistema mostrará un mensaje de aviso indicando al usuario que debe rellenar dichos campos.

Si todo ha ido correctamente, el sistema indicará al usuario que la acción se ha dado de alta correctamente tal y como se puede ver en la Ilustración 109. Por lo tanto, se creará una nueva entrada en el fichero Acciones.txt.

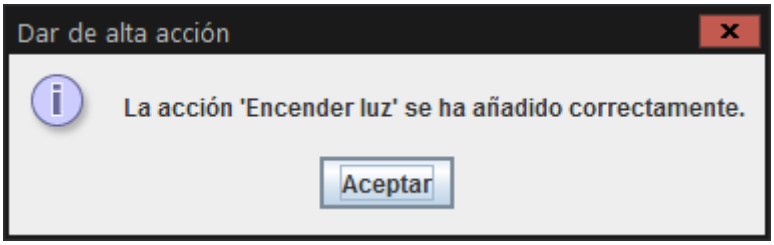

Ilustración 109: Mensaje Alta acción

Puede ocurrir que la acción que se va a dar de alta ya exista en el fichero Acciones.txt, en este caso el sistema informará de ello con el mensaje siguiente.

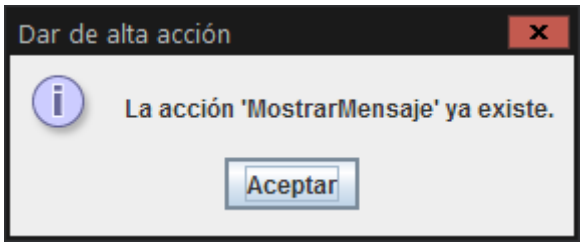

Ilustración 110: Mensaje de información

H.12 Ventana Baja acción

La ventana que se observa en la Ilustración 111 corresponde a la ventana en la que se da de baja una acción correspondiente a un periférico. Cada vez que se seleccione el periférico se cargaran automáticamente las acciones correspondientes a dicho periférico en el campo "Acción".

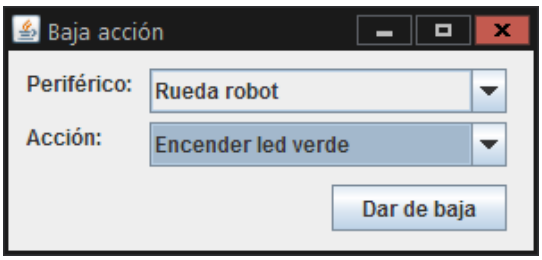

Ilustración 111: Baja acción

En el instante en el que el usuario pulsa sobre el botón "Dar de baja" el sistema eliminará del fichero Acciones.txt la línea correspondiente al periférico y la acción e informará de ello al usuario (Ilustración 112).

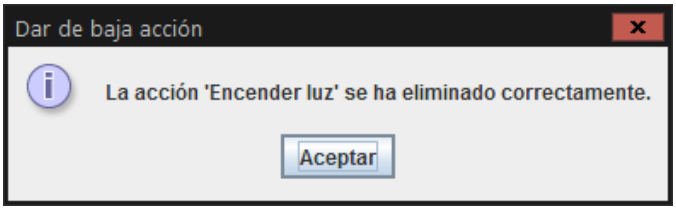

Ilustración 112: Mensaje Baja acción

I Tabla comandos

Función	*Interfaz gráfica*		*Robot*	
	Enviado	**Recibido**	**Recibido**	**Enviado**
Comprobar conexión	-C	Conectado	-C	Conectado
Girar robot a izquierda	-I	OK	-I	OK
Girar robot a derecha	-D	OK	-D	OK
Rueda Izquierda: Girar izquierda	-A	OK	-A	OK
Rueda Izquierda: Girar derecha	-B	OK	-B	OK
Rueda Derecha: Girar izquierda	-D	OK	-D	OK
Rueda Derecha: Girar derecha	-E	OK	-E	OK
Avanzar robot	-F	OK	-F	OK
Retroceder robot	-G	OK	-G	OK
Encender led rojo	-H	Encendido	-H	Encendido
Apagar led rojo	-J	Apagado	-J	Apagado
Encender led verde	-K	Encendido	-K	Encendido
Apagar led verde	-L	Apagado	-L	Apagado
Encender led rojo tricolor	ER	Encendido	ER	Encendido
Apagar led rojo tricolor	AR	Apagado	AR	Apagado
Encender led verde tricolor	EV	Encendido	EV	Encendido
Apagar led verde tricolor	AV	Apagado	AV	Apagado
Encender led azul tricolor	EA	Encendido	EA	Encendido
Apagar led azul tricolor	AA	Apagado	AA	Apagado
Detectar distancia	DD	[Distancia]	DD	[Distancia]
Detector de luz	DL	[Umbral luz]	DL	[Umbral luz]
Detector de sonido	DS	[Umbral sonido]	DS	[Umbral sonido]
Detector de temperatura	DT	[Temperatura]	DT	[Temperaura]
Detector de contacto físico Izquierdo	CI	SI/NO	CI	SI/NO
Detector de contacto físico Derecho	CD	SI/NO	CD	SI/NO
Zumbador bajo	ZB	--	ZB	--
Zumbador medio	ZM	--	ZM	--
Zumbador alto	ZA	--	ZA	--
Seguidor de línea izquierdo	SI	Blanco/Negro	SI	Blanco/Negro
Seguidor de línea	SD	Blanco/Negro	SD	Blanco/Negro

derecho				
Pulsador 1	P1	Pulsado	P1	Pulsado
Pulsador 2	P2	Pulsado	P2	Pulsado
Pantalla LCD – Enviar dato	PL	--	PL	--
Acelerómetro	AC	[Rango]	AC	[Rango]
Altímetro	AL	[Metro]	AL	[Metros]
Brújula	BR	[Grados]	BR	[Grados]
Giroscopio	GI	[º/s]	GI	[º/s]

Tabla 2: Comandos PC-Robot

J Pruebas de caja negra

Detectar puerto de comunicación:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Cargar periféricos".	El sistema muestra los puertos de comunicación en una nueva ventana.	Corregido. El sistema no cargaba los puertos de comunicación conectados al pc.

Tabla 3: Detectar puerto de comunicación

Leer y cargar los periféricos del robot:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Aceptar" sin haber seleccionado el puerto.	El sistema solicita que seleccione un puerto de comunicación.	Correcto.
El usuario pulsa sobre el botón "Aceptar" una vez haya seleccionado el puerto.	El sistema carga los periféricos en la interfaz gráfica.	Correcto.
El usuario pulsa sobre el botón "Cancelar".	El sistema no realiza ninguna acción.	Correcto.

Tabla 4: Leer y cargar los periféricos del robot

Ver el número máximo de cada periférico:

Prueba	Resultado Obtenido	Valoración
Tras cargar los periféricos la aplicación debe mostrar el periférico y su número máximo en un área de texto.	El sistema muestra esta información.	Corregido. El sistema no mostraba correctamente esta información.

Tabla 5: Ver el número máximo de cada periférico

Ver detalle de un periférico:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Ver detalles" sin haber seleccionado un periférico.	El sistema solicita que seleccione un periférico.	Correcto.
El usuario pulsa sobre el botón "Ver detalles" habiendo seleccionado un periférico.	El sistema muestra la descripción del periférico.	Correcto.

Tabla 6: Ver detalle de un periférico

Testeo de los periféricos:

Prueba	Resultado Obtenido	Valoración
El robot no está conectado y el usuario pulsa sobre el botón "Test periféricos".	El sistema solicita conecte el robot al pc.	Correcto.
El robot está conectado y el usuario pulsa sobre el botón "Test periféricos".	El sistema inicia la ventana para testear los periféricos.	Correcto.

Tabla 7: Testeo de los periféricos

Añadir periférico:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Añadir periférico" sin haber seleccionado ningún periférico.	El sistema solicita que seleccione un periférico.	Correcto.
El usuario pulsa sobre el botón "Añadir periférico" habiendo seleccionado un periférico.	El sistema añade el periférico al entorno de trabajo.	Correcto.

Tabla 8: Añadir periférico

Eliminar periférico:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Eliminar periférico".	El sistema elimina el periférico del entorno de trabajo.	Corregido. El sistema no actualizaba el periférico eliminado.

Tabla 9: Eliminar periférico

Eliminar todos periféricos:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Eliminar todo".	El sistema elimina todos los periféricos del entorno de trabajo.	Corregido. El sistema no actualizaba los periféricos eliminados

Tabla 10: Eliminar todos periféricos

Cargar situaciones de cada periférico:

Prueba	Resultado Obtenido	Valoración
Sin periféricos en el entorno de trabajo, el usuario pulsa sobre el botón "Definir funcionalidades".	El sistema indica al usuario que añada periféricos al entorno de trabajo.	Correcto.
Diseño sin rueda en el entorno de trabajo, el usuario pulsa sobre el botón "Definir funcionalidades".	El sistema indica al usuario que el diseño tiene que tener las dos ruedas.	Correcto.
Con periféricos en el entorno de trabajo, el usuario pulsa sobre el botón "Definir funcionalidades".	El sistema inicia una ventana con todas las situaciones de cada periférico presentes en el robot.	Correcto.

Tabla 11: Cargar situaciones de cada periférico

Seleccionar situaciones de cada periférico:

Prueba	Resultado Obtenido	Valoración
El usuario selecciona una situación	El sistema marca la situación seleccionada.	Correcto.

Tabla 12: Seleccionar situaciones de cada periférico

Cargar acciones de cada periférico:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre una situación.	El sistema abrirá una ventana para seleccionar las acciones que se llevará a cabo en tal situación.	Corregido. El sistema no mostraba las acciones de los periféricos.

Tabla 13: Cargar acciones de cada periférico

Seleccionar acciones de cada periférico:

Prueba	Resultado Obtenido	Valoración
El usuario chequea las acciones de los periféricos según su necesidad.	El sistema almacenará los valores chequeados.	Correcto.

Tabla 14: Seleccionar acciones de cada periférico:

Mostrar funcionalidades:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Aceptar" que hay en la ventana "Seleccionar acciones".	El sistema actualizará el panel izquierdo de la ventana "Definir funcionalidades" con un resumen de las acciones definidas.	Corregido. El sistema no actualizaba este panel debido a que había un error.

Tabla 15: Mostrar funcionalidades

Crear programa:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Cargar funciones al robot".	El sistema creará el programa main.c con todas las funcionalidades definidas por el usuario en el diseño, lo compilara para obtener el .hex y lo cargará al robot.	Corregido. El sistema no generaba el fichero .hex debido a que el comando utilizado para ello daba un error.

Tabla 16: Crear programa

Cargar a Robot:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Cargar funciones al robot".	El sistema cargará el programa .hex al robot.	Corregido. Había un error al generar el fichero .hex.

Tabla 17: Cargar a Robot

Guardar diseño:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Guardar/Guardar como..".	El sistema guarda el diseño elaborado en el entorno gráfico.	Correcto.

Tabla 18: Guardar diseño:

Abrir diseño:

Prueba	Resultado Obtenido	Valoración
El usuario pulsa sobre el botón "Abrir".	El sistema abre un diseño guardado previamente.	Correcto.

Tabla 19: Abrir diseño

Dar de alta un periférico:

Prueba	Resultado Obtenido	Valoración
El usuario da de alta un periférico que ya existe.	El sistema indica que el periférico ya existe.	Correcto.
El usuario da de alta un periférico que no existe.	El sistema da de alta el periférico y lo añade al fichero Periféricos.txt.	Correcto.

Tabla 20: Dar de alta un periférico

Dar de baja un periférico:

Prueba	Resultado Obtenido	Valoración
El usuario da de baja un periférico.	El sistema da de baja el periférico y lo elimina del fichero Periféricos.txt.	Corregido. El sistema no eliminaba el periférico del fichero.

Tabla 21: Dar de baja un periférico

Dar de alta una acción:

Prueba	Resultado Obtenido	Valoración
El usuario da de alta una acción que ya existe.	El sistema indica que la acción ya existe.	Correcto.
El usuario da de alta una acción que no existe.	El sistema da de alta la acción y lo añade al fichero Acciones.txt.	Correcto.

Tabla 22: Dar de alta una acción

Dar de baja una acción:

Prueba	Resultado Obtenido	Valoración
El usuario da de baja una acción.	El sistema da de baja la acción y lo elimina del fichero Acciones.txt.	Correcto.

Tabla 23: Dar de baja una acción

Dar de alta una situación:

Prueba	Resultado Obtenido	Valoración
El usuario da de alta una situación que ya existe.	El sistema indica que la situación ya existe.	Correcto.
El usuario da de alta una situación que no existe.	El sistema da de alta la situación y lo añade al fichero Situaciones.txt.	Correcto.

Tabla 24: Dar de alta una situación

Dar de baja una acción:

Prueba	Resultado Obtenido	Valoración
El usuario da de baja una situación.	El sistema da de baja la situación y lo elimina del fichero Situaciones.txt.	Correcto.

Tabla 25: Dar de baja una acción

K Pruebas del sistema completo

Prueba 1:

En primer lugar se deberá de conectar la placa al ordenador a través del puerto USB. Iniciamos la aplicación y observamos que cuando pulsamos sobre el botón "Cargar periféricos" esta la reconoce. Cuando el usuario selecciona el puerto y pulsa sobre el botón "Aceptar" los periféricos se cargan en la interfaz.

Posteriormente se van añadiendo al entorno gráfico los periféricos según nuestra necesidad. Para realizar pruebas del sistema completo se ha utilizado los siguientes periféricos: ruedas, led rojo, led verde, pulsador 1, pulsador 2 y sensor de luz (Ver Ilustración 114). A la placa se ha añadido un circuito para poder usar el sensor de luz tal y como se puede ver en la Ilustración 113.

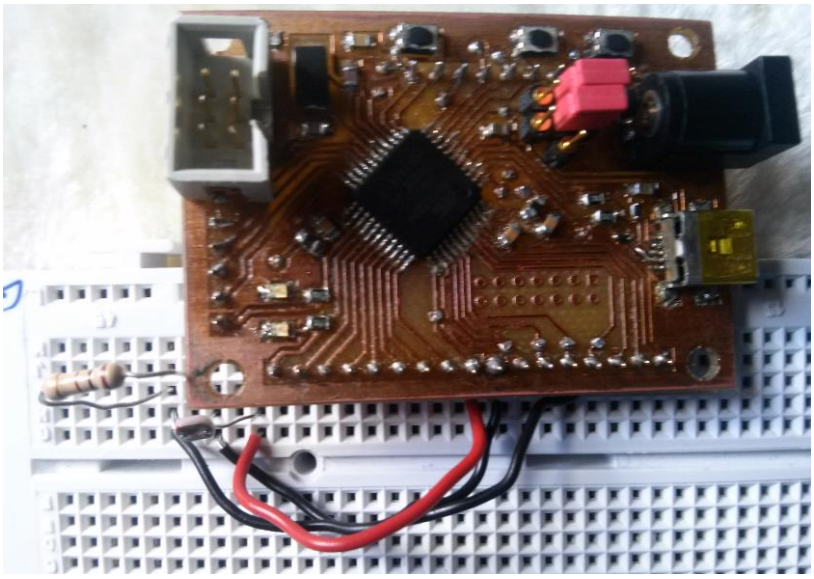

Ilustración 113: Circuito Sensor de luz

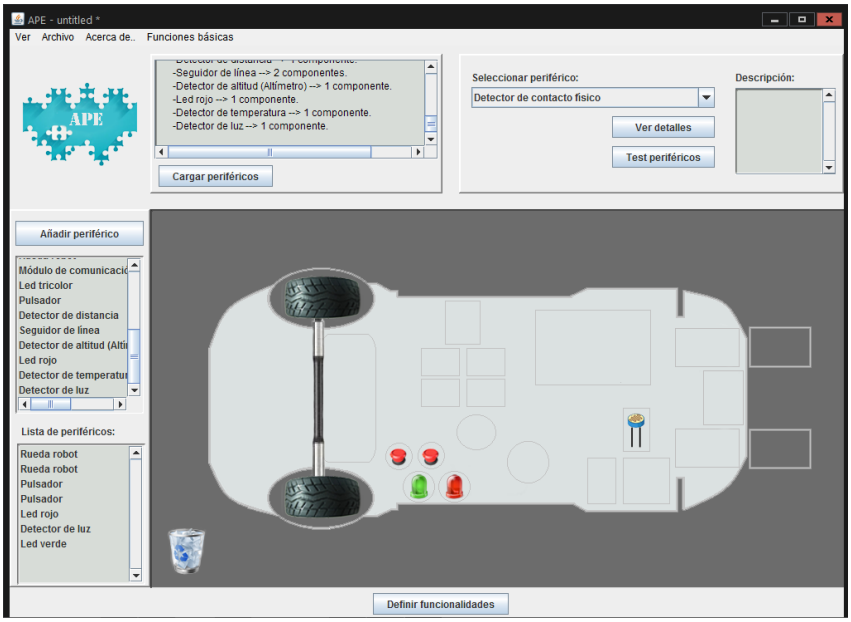

Ilustración 114: Periféricos en el entorno gráfico

Cuando se presiona sobre el botón “Definir funcionalidades” la aplicación nos mostrará las situaciones en la que se puede encontrar el robot según los periféricos disponibles en el diseño.

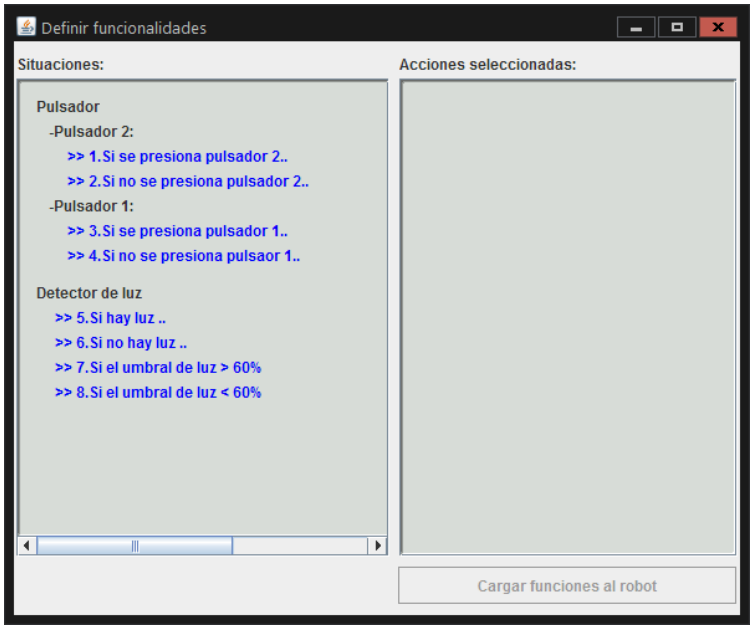

Ilustración 115: Funcionalidades del robot

Se puede ver que la aplicación ha cargado correctamente solo las situaciones correspondientes a los periféricos que se encuentran en el diseño elaborado en el entorno gráfico. No hay situaciones correspondientes a los leds ni ruedas debido a que en el fichero Situaciones.txt no se ha definido ninguna situación para estos periféricos. A continuación se van a definir las acciones que se desea que realice el robot en las situaciones que deseadas.

Cuando se pulsa sobre la situación ***"5.Si hay Luz",*** se abrirá una nueva ventana para seleccionar las acciones que se desea que se realice en este caso. Para esta acción seleccionamos ***"Encender led rojo"*** y ***"Apagar led verde":***

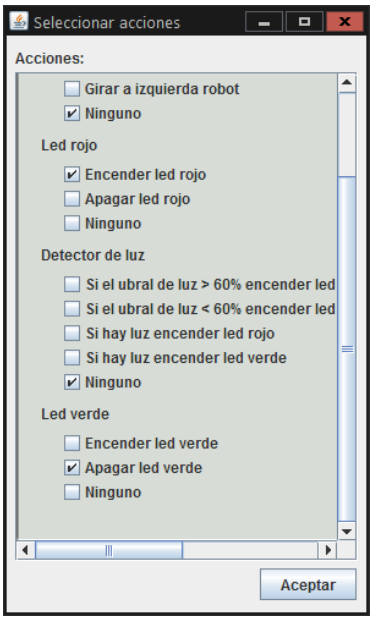

Ilustración 116: Opción Encender Led rojo y apagar led verde

Tras pulsar sobre el botón "Aceptar" vemos como se actualiza el panel de la derecha con el resumen de las funciones que se van definiendo.

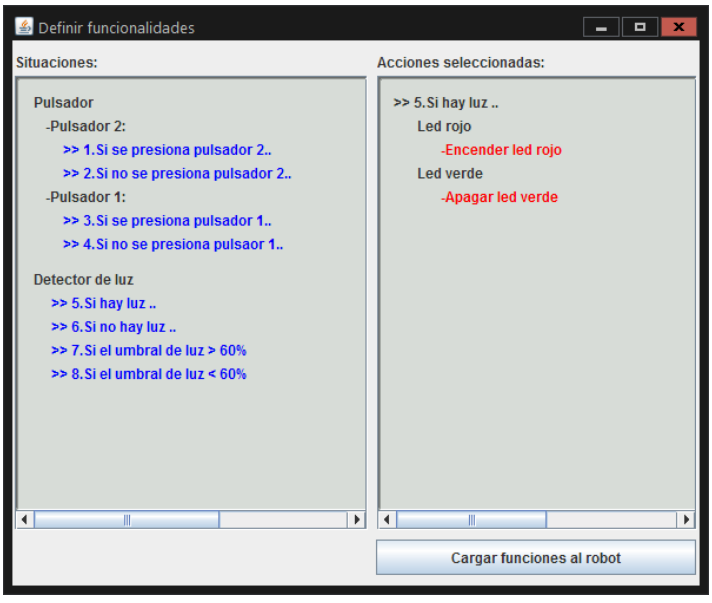

Ilustración 117: Actualiza panel resumen

Ahora se va a seleccionar acciones para la situación ***"6.Si no hay luz",*** se puede ver que la opción encender led rojo y apagar led verde ya no se pueden seleccionar dado

que ya han sido seleccionado anteriormente. En este caso se va a seleccionar apagar led rojo y encender led verde.

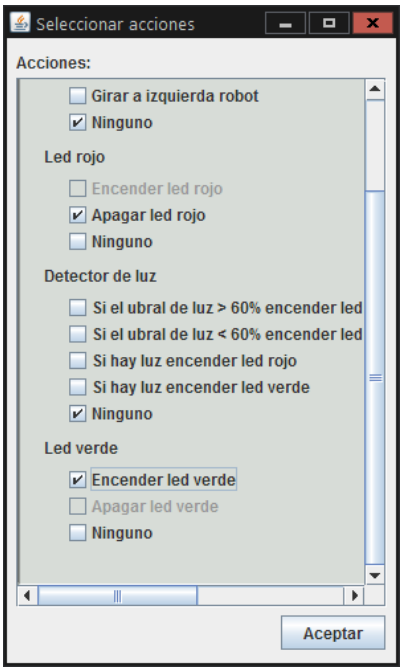

Ilustración 118: Acciones ya seleccionadas

A continuación se procede a cargar estas funcionalidades al robot pulsando sobre el botón “Cargar funciones al robot”.

El programa main.c que genera el sistema se ve en la Ilustración 119.

```
#include "Rutinas.h"

int main(void)
{
    wdt_disable();

    inicializacion();

    while(1)

    {

        if(hayLuz())
        {
            encenderLedRojo();

            apagarLedVerde();

        }

        if(!hayLuz())
        {
            encenderLedVerde();

            apagarLedRojo();

        }

    }
}
```

Ilustración 119: Programa mai.c que genera el sistema

El resultado final se puede ver en la Ilustración 121, cuando la luz incide sobre el sensor, el led rojo se enciende y el led verde se apaga.

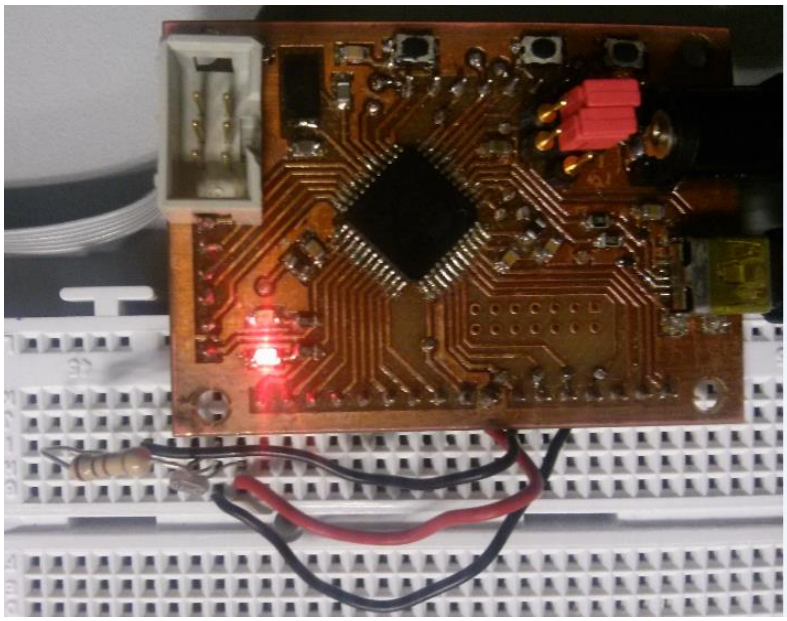

Ilustración 120: Led rojo encendido

En la Ilustración 122 se ve que cuando no incide luz sobre el sensor el led rojo se apaga y el led verde se enciende.

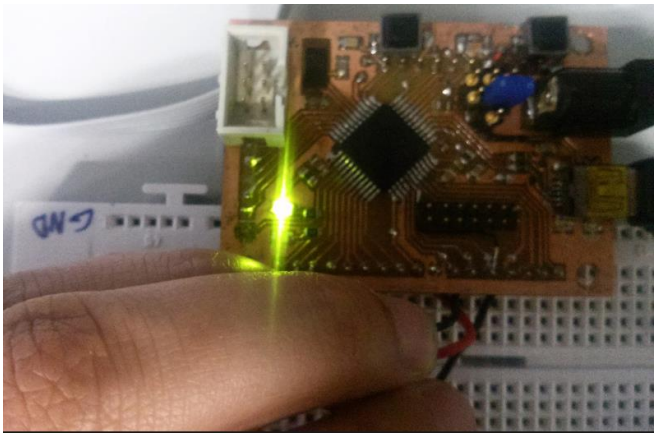

Ilustración 121: Led verde encendido

Prueba 2:

En esta prueba se selecciona la situación ***"1.Si se presiona pulsador 2"*** se va a realizar la acción ***"Si el umbral de luz < 60% encender led verde"*** y ***"2.Si no se presiona sobre el pulsador 2"*** se realiza la acción ***"Si el umbral de luz > 60% encender led rojo".***

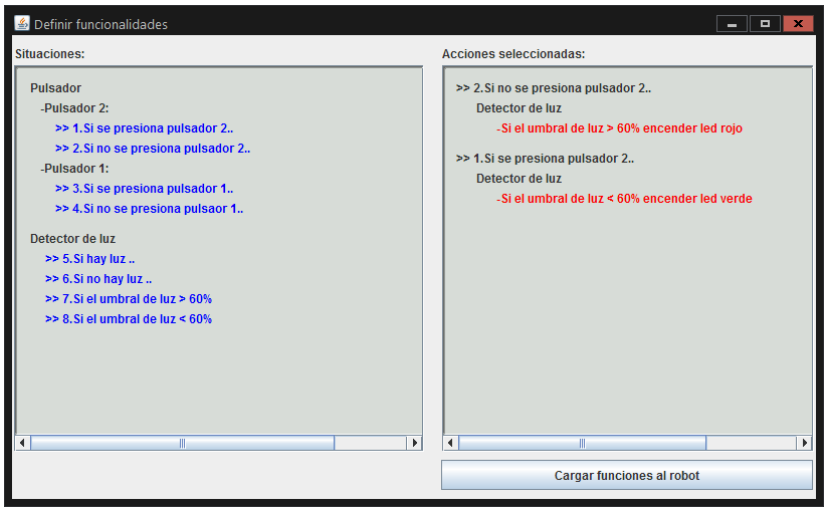

Ilustración 122: Prueba 2

El programa main.c que genera el sistema se ve en la Ilustración 123.

```
#include "Rutinas.h"

int main(void)
{
    wdt_disable();

    inicializacion();

    while(1)

    {

        if(pulsador2Pulsado())
        {
            encenderLedVerdeUmbralLuzMenorA(60);

        }

        if(!pulsador2Pulsado())
        {
            encenderLedRojoUmbralLuzMayorA(60);

        }

    }
}
```

Ilustración 123: Programa mai.c que genera el sistema

En la Ilustración 124 se ve que cuando no está presionado el botón 2 y el umbral de luz es mayor que el 60% se enciende el led rojo.

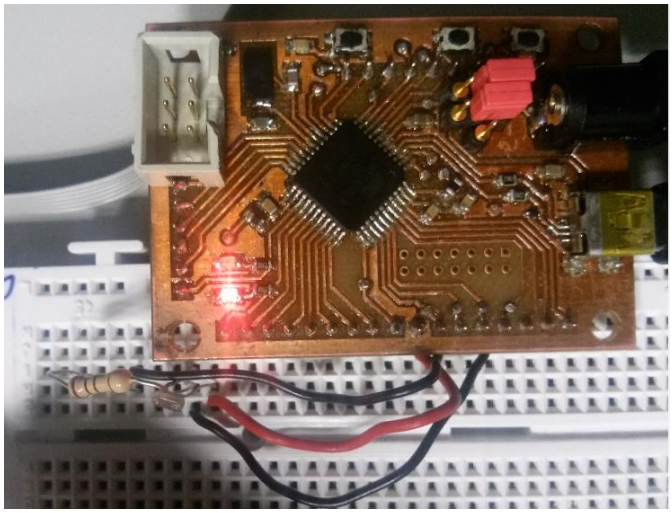

Ilustración 124: Pulsador 2 no presionado → Led rojo encendido

En la Ilustración 125 se ve que cuando está presionado el botón 2 y el umbral de luz es menor que el 60% se enciende el led verde.

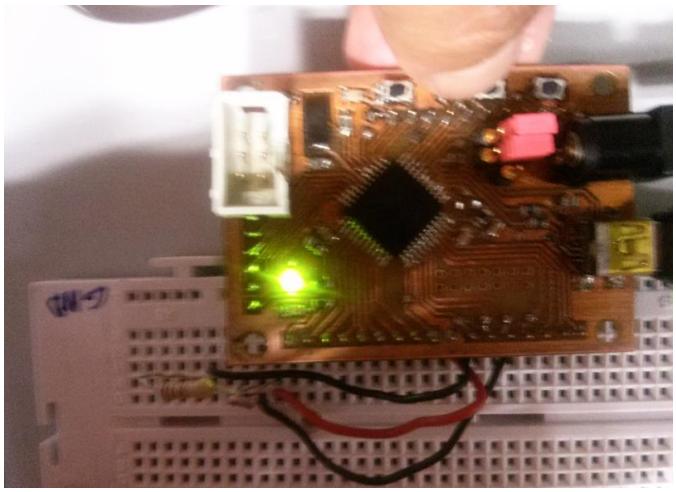

Ilustración 125 : Pulsador 2 presionado → Led verde encendido

Prueba 3

En esta prueba se comprueba la ventana de test de periféricos. Se comprobarán los leds, los pulsadores y el sensor de luz.

Leds:
Se pulsa sobre el botón encender led rojo, con lo cual se envía un comando al robot para encender este led y el resultado es satisfactorio. Lo mismo sucede con el led verde.

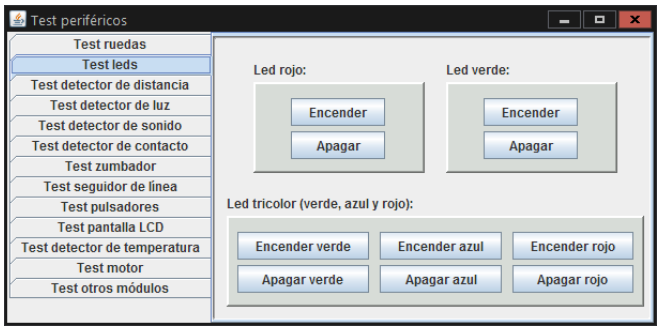

Ilustración 126: Test encender led rojo

Pulsadores:
Se pulsa sobre el botón 2 enviando así a la interfaz gráfica el comando correspondiente a esta acción, el sistema lo interpreta y modifica el texto y la imagen del pulsador.

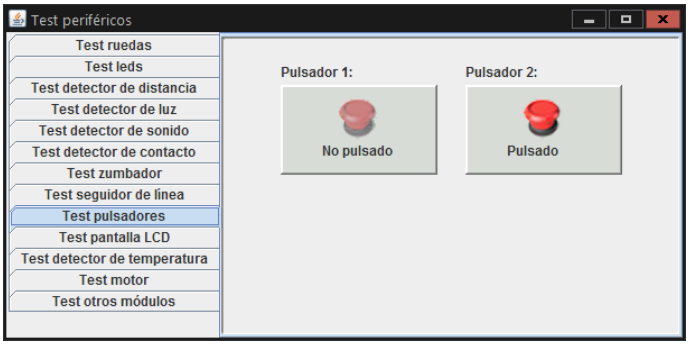

Ilustración 127: Test Pulsador 2

Sensor de luz:

En este caso la aplicación muestra el umbral de luz que incide sobre este sensor.

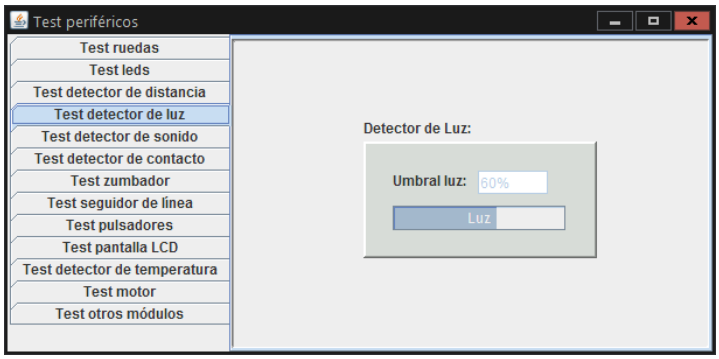

Ilustración 128: Test Sensor de luz

Dado que no se ha llegado a implementar el resto de funciones para los demás periféricos no se han podido ejecutar pruebas sobre ellos. Aunque con las pruebas expuestas se puede comprobar que el sistema funciona correctamente y cumple con el objetivo planteado en este trabajo.

L Pruebas de caja blanca

Unción	Descripción	Valoración
checkSerialPort()	Comprueba si hay puertos conectados al pc y si existe el puerto de conexión al robot. Carga el programa VirtualSerial.hex.	Corregido. No reconocía todos los puertos conectados al pc.
testPortComunication()	Envía datos a los puertos de comunicación para comprobar si es el puerto correcto.	Corregido. La recepción de datos procedentes del robot no se realizaba de forma correcta.
programMicroProtComunication()	Carga el programa VirtualSerial.hex en el robot.	Corregido. No se ejecutaba correctamente el comando usado para cargar el programa al robot. También se hacen control de errores por si hay algún error durante la ejecución de estos comandos.
createSetencias()	Crea las sentencias correspondientes a las funciones que ha definido el usuario en la aplicación.	Corregido. No se creaba las sentencias correctamente ya que o se leían correctamente las funciones definidas por el usuario.
createProgram()	Crea el programa main.c	Correcto.
getFileHEX()	Crea el programa hexadecimal correspondiente al programa main.c.	Correcto.
checkFileHEX()	Comprueba si el programa hexadecimal correspondiente al main.c se ha creado.	Correcto.
loadHEXToMicro()	Carga el programa Hexadecimal correspondiente al main.c al robot.	Corregido. No se ejecutaba correctamente el comando usado para cargar el programa al robot. También se hacen control de errores por si hay algún error durante la ejecución de estos comandos.
loadProgramInRobot()	Crea el programa main.c, obtiene el programa hexadecimal, comprueba si se ha creado y lo carga al robot.	Correcto. Tras corregir las funcionas que hace uso.
openTestPeripheralsFrame()	Comprueba si hay conexión con el robot y si lo hay abre la ventana para testear los periféricos.	Corregido. En caso de no haber conexión, no se mostraba la

		posibilidad de seleccionar nuevamente el puerto.
startTestPeripheralsFrame()	Inicia la ventana para testear los periféricos.	Correcto.
startSelectionSerialPortFrame()	Inicia la ventana para seleccionar el puerto de comunicación.	Correcto.
openSelectActionFrame()	Abre la ventana para definir las funciones que va a tener el robot.	Correcto.
startSelectActionRobotFrame()	Inicia la ventana para definir las funciones que va a tener el robot.	Correcto.
setResizableComponents()	Reposiciona los componentes de la ventana donde se define las funciones que tendrá el robot cuando se redimensiona.	Corregido. Los componentes se descuadraban en esta ventana.
addComponentListener()	Añade un listener cuando se cambie el tamaño de la ventana donde se definen las funciones que tendrá el robot.	Correcto.
createNewPeripheral()	Crea un nuevo periférico.	Correcto.
addListPeripherals()	Añade un periférico en la parte superior del entono de trabajo.	Corregido. Si se añadían varios periféricos y no cabían en la parte superior, estos desaparecían.
createPeripherals()	Crea y añade un periférico al entorno de trabajo.	Corregido. Tras corregir la función ***addListPeripherals().***
isListPeripheralsComplete()	Comprueba si hay espacio en la parte superior del entorno de trabajo para seguir introduciendo periféricos.	Correcto.
readRobotPeripherals()	Comprueba si hay conexión con el robot y si lo hay inicia la ventana para seleccionar el puerto de comunicación para posteriormente cargar los periféricos en la interfaz.	Correcto.
updateNumPeripheral()	Actualiza el número de periféricos presentes en el entorno de trabajo.	Corregido. Cuando se eliminaba un periférico no se actualizaba correctamente.
deletePeripheralsWorkspace()	Elimina un periférico del entorno de trabajo.	Correcto.
updateAllNumPeripheral()	Actualiza todos los periféricos presentes en el entorno de trabajo.	Corregido. Cuando se eliminaban todos los periféricos no se actualizaba correctamente.
deleteAllPeripheralsWorkspace()	Elimina todos los periféricos del entorno de trabajo.	Correcto.
viewDetailsPeripheral()	Muestra los detalles de un periférico.	Corregido. No se controlabamos si no se ha seleccionado un

		periférico.
setNumMaxPeripherals()	Asigna el número máximo de cada periférico en el robot.	Correcto.
setNumPeripherals()	Asigna el número de los periféricos que se encuentran en el robot.	Corregido.
readPeripheralsInRobot()	Lee la lista de periféricos presente en el robot.	Correcto.
startAddActionFrame()	Inicia la ventana para dar de alta una acción correspondiente a un periférico.	Correcto.
startAddSituationFrame()	Inicia la ventana para dar de alta una situación correspondiente a un periférico.	Correcto.
startAddPeripheralFrame()	Inicia la ventana para dar de alta un periférico.	Correcto.
startDeleteActionFrame()	Inicia la ventana para dar de baja una acción correspondiente a un periférico.	Correcto.
startDeleteSituationFrame ()	Inicia la ventana para dar de baja una situación correspondiente a un periférico.	Correcto.
startDeletePeripheralFrame()	Inicia la ventana para baja un periférico.	Correcto.
startActionsRobotFrame()	Inicia la ventana para seleccionar una acción.	Correcto.
addListenerSituations()	Listener para abrir la ventana con las acciones de cada periférico. Se ejecuta cuando se selecciona una situación de la ventana donde definimos las funcionalidades.	Correcto.
peripheralsSituationsInRobot()	Lee los periféricos correspondientes a cada situación.	Correcto
viewSituationsInPanel()	Muestra las situaciones en la ventana donde se definen las funcionalidades que tendrá el robot.	Corregido. Las situaciones no se mostraban correctamente ya que solo se debían mostrar aquellas correspondientes a los periféricos disponibles en el robot.
readSituationOfRobot()	Lee las situaciones de cada periférico presente en el robot y las muestra en el panel.	Corregido. Correcto tras corregir la función ***viewSituationsInPanel().***
readActionsRobot()	Lee las acciones que tendrá el robot dependiendo de los periféricos añadidos al robot.	Corregido. No se leían correctamente las acciones correspondientes a cada periférico.
createJLabelsSituationsInFrame()	Para cada situación se crea un JLabel.	Correcto.
deregisterSituationsInFile()	Da de baja una situación y la elimina del fichero Situaciones.txt.	Correcto.
saveFileSituations()	Elimina la situación del fichero Situaciones.txt y lo guarda.	Correcto.
readFileListSituations()	Lee el fichero Situaciones.txt con las situaciones de cada	Correcto.

	periférico presentes en el robot.	
registerSituationInFile()	Da de alta una situación en el fichero Situaciones.txt.	Correcto.
addSituation()	Añade la situación al fichero Situaciones.txt.	Correcto.
registerPeripheralInFile()	Da de alta un periférico en el fichero Periféricos.txt.	Correcto.
addPeripheral()	Añade un periférico al fichero Periféricos.txt.	Correcto.
deregisterPeripheralInFile()	Da de baja un periférico y la elimina del fichero Periféricos.txt.	Correcto.
saveFilePeripherals()	Elimina el periférico del fichero Periférico.txt y lo guarda.	Correcto.
readFileListPeripherals()	Lee el fichero Periféricos.txt con los periféricos presentes en el robot.	Correcto.
deregisterActionInFile()	Da de baja una acción y la elimina del fichero Acciones.txt.	Correcto.
saveFileActions()	Elimina la acción del fichero Acciones.txt y lo guarda.	Correcto.
readFileListActions()	Lee el fichero Acciones.txt con las acciones presentes en el robot.	Correcto.
registerActionInFile()	Da de alta una acción en el fichero Acciones.txt.	Correcto
addAction()	Añade la acción al fichero Acciones.txt.	Correcto.
readActionsRobot()	Lee el fichero Acciones.txt con las acciones de cada periférico presentes en el robot.	Correcto.
createCheckBoxOptions	Para cada acción se crea un CheckBox.	Correcto
updatePanelActionsRobot()	Actualiza un panel con un resumen de las acciones.	Corregido. No actualizaba la lista correctamente.
doSaveAPEFile()	Guarda el diseño.	Correcto.
doSaveAsAPEFile()	Guardar como.. el diseño.	Correcto.
doOpenAPEFile()	Abre un diseño guardado anteriormente.	Correcto.

Tabla 26: Pruebas de caja blanca